Chapitre 9. Algorithmes de tri et algorithmes gloutons

Table des matières

Remplissez le jupyter notebook suivant en vous aidant de votre livre de Première NSI de Serge BAYS .

- Pour répondre, double-cliquez sur **Réponse** et complétez la zone en-dessous. Puis cliquez sur le bouton *Exécuter*.
- **Important : pour fermer votre jupyter notebook, cliquez sur :**

Fichier / Créer une nouvelle sauvegarde

puis sur :

Fichier / Fermer et Arrêter

- Ecrivez ci-dessous votre prénom et votre nom :

Réponse :

Chapitre 9. Algorithmes de tri et algorithmes gloutons

1. Les algorithmes de tri

Lisez l'introduction du chapitre p. 307

1) En gestion informatique des données, on trie les données. Qu'est-ce que cela permet ?

Réponse :

2) Durant la deuxième guerre mondiale, une des programmatrices du premier ordinateur entièrement électronique (l'ENIAC) a développé à Philadelphie le premier algorithme de tri. Quel est son nom ?

Réponse :

1.1 Introduction

Lisez le paragraphe **Introduction** p. 309 au milieu de la p. 310

3) Sur quels langages de programmation ont travaillé les informaticiennes Betty HOLBERTON et Grace HOPPER au milieu du 20e siècle ?

Réponse :

4) Dans les algorithmes vus plus loin :

1. Tri par sélection
2. Tri par insertion

Quels sont les deux types d'opérations que nous effectuerons sur les éléments à trier ?

Réponse :

5) La complexité d'un algorithme s'exprime en ordre de grandeur O du nombre nombre d'opérations à faire pour traiter une liste de n éléments. Dans le cas d'un algorithme de tri par sélection ou de tri par insertion, de quelles opérations s'agit-il ?

Réponse :

6) A quel moment faut-il absolument évaluer le coût temporel d'un algorithme ?

Réponse :

Nombre de façons de ranger (on dit aussi de permuter) 3 éléments a, b, c :

- Pour le choix du 1 er élément, il y a 3 possibilités.
- Pour le choix du 2 e élément, il reste 2 possibilités.
- Pour le dernier élément, il reste 1 possibilité.

Soit au total $3 \times 2 \times 1 = 6$ rangements possibles (on dit aussi nombre de permutations).

Voici un tableau montrant les 6 permutations de 3 éléments :

Si le 1er élément est a	Si le 1er élément est b	Si le 1er élément est c
(a, b, c)	(b, a, c)	(c, a, b)
(a, c, b)	(b, c, a)	(c, b, a)

Pour 3 éléments, il y a bien $r = 3 \times 2 \times 1 = 6$ rangements possibles.

Le nombre $3 \times 2 \times 1 = 6$ est appelé *factorielle 3* et se note $3!$

7) Quel est le nombre de manières r possibles de ranger 4 éléments (a, b, c, d) ?

Réponse :

Voici un tableau montrant toutes les permutations de 4 éléments :

Si le 1er élément est a	Si le 1er élément est b	Si le 1er élément est c	Si le 1er élément est d
(a, b, c, d)	(b, a, c, d)	(c, a, b, d)	(d, a, b, c)
(a, b, d, c)	(b, a, d, c)	(c, a, d, b)	(d, a, c, b)
(a, c, b, d)	(b, c, a, d)	(c, b, a, d)	(d, b, a, c)
(a, c, d, b)	(b, c, d, a)	(c, b, d, a)	(d, b, c, a)
(a, d, b, c)	(b, d, a, c)	(c, d, a, b)	(d, c, a, b)
(a, d, c, b)	(b, d, c, a)	(c, d, b, a)	(d, c, b, a)

On retient :

- Le nombre de manières r possibles de ranger n éléments est :

$$r = n \times (n-1) \times \ldots \times 2 \times 1 = n!.$$

Attention : il manque les pointillés dans le livre au bas de la p. 309 pour cette formule. Les pointillés signifient "le produit de tous les entiers de n à 1".

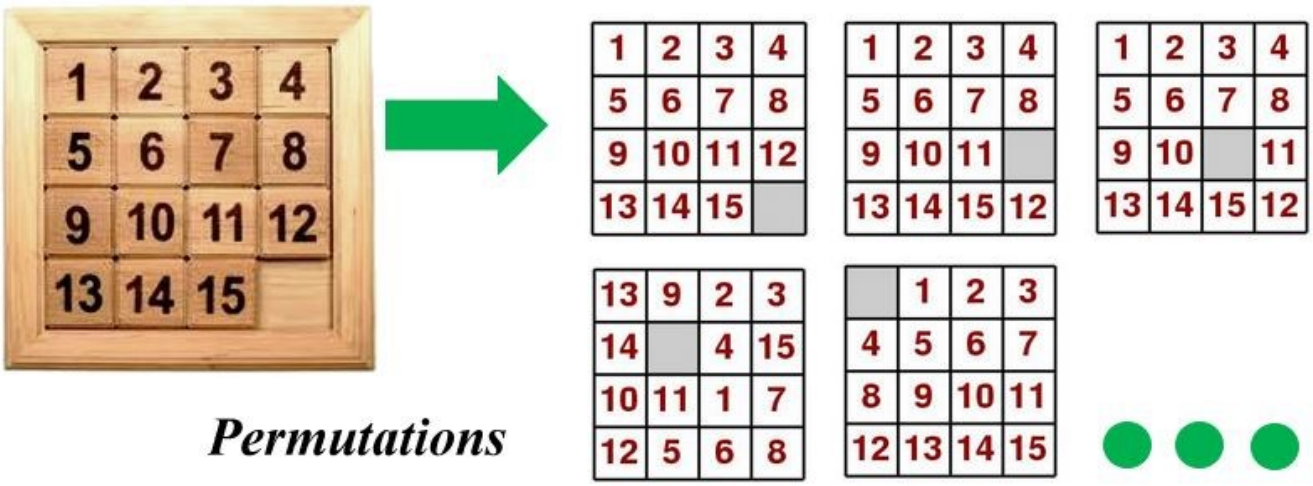
8) Quel est le nombre de manières r possibles de permuter 69 éléments ?

Indication : vous pouvez utiliser la calculatrice pour calculer factorielle 69 :

- Saisir 69
- Touche math
- Menu PROB
- Symbole **!**

Réponse :

- Exemples de permutations de 16 éléments avec le jeu de taquin :



Combien de comparaisons faut-il faire pour *sélectionner un rangement parmi* tous ceux possibles ?

Reprenons les $3! = 6$ rangements possibles de 3 éléments a, b, c. On veut sélectionner parmi ces 6, le rangement (a, b, c).

- Une première comparaison $a < b$ (qui signifie "a est placé avant b") permet d'éliminer la moitié des rangements. Il en reste trois.

Le 1er élément est a	Le 1er élément est b	Le 1er élément est c
(a, b, c) •	(b, a, c)	(c, a, b) •
(a, c, b) •	(b, c, a)	(c, b, a)

- Une deuxième comparaison $c < a$ permet d'éliminer la moitié des trois rangements restants.

Si $c < a$ est vrai (premier tableau ci-dessous) alors il ne reste qu'un rangement sélectionné et donc le tri est terminé.

Si $c < a$ est faux (deuxième tableau ci-dessous) alors il reste deux rangements.

1er élément est a	1er élément est b	1er élément est c
(a, b, c) •	(b, a, c)	(c, a, b) • •
(a, c, b) •	(b, c, a)	(c, b, a)

1er élément est a	1er élément est b	1er élément est c
(a, b, c) • •	(b, a, c)	(c, a, b) •
(a, c, b) • •	(b, c, a)	(c, b, a)

- Dans le cas où il reste deux rangements, il faut une troisième comparaison, par exemple on teste si $b < c$.

Le tableau ci-dessous montre le cas où $b < c$ est vrai.

Le 1er élément est a	Le 1er élément est b	Le 1er élément est c
(a, b, c) • • •	(b, a, c)	(c, a, b)
(a, c, b)	(b, c, a)	(c, b, a)

En résumé :

- Il y a $3! = 6$ rangements possibles de 3 éléments.
- 2 tests successifs permettent de sélectionner un rangement parmi $2^2 = 4$ rangements.
- 3 tests successifs permettent de sélectionner un rangement parmi $2^3 = 8$ rangements.

On a $2^3 \geq 3!$, donc dans le pire cas, il faut 3 comparaisons pour sélectionner un parmi 6 rangements de 3 éléments.

9) Rangements (ou permutations) de 4 éléments : Complétez les pointillés dans la cellule ci-dessous :

Réponse :

- Il y a $4! = \ldots$ rangements possibles de 4 éléments.
- 2 tests successifs permettent de sélectionner un rangement parmi $2^2 = 4$ rangements.
- 3 tests successifs permettent de sélectionner un rangement parmi $2^3 = 8$ rangements.
- 4 tests successifs permettent de sélectionner un rangement parmi $2^4 = \ldots$ rangements.
- 5 tests successifs permettent de sélectionner un rangement parmi $2^5 = \ldots$ rangements.

On a $2^{\cdots} \geq 4!$, donc dans le pire cas, il faut ... comparaisons pour sélectionner un parmi ... rangements de 4 éléments.

10) Combien y a-t-il de permutations possibles pour 10 cartes à jouer ?

Réponse :

11) En envisageant le pire cas possible, combien de comparaisons faut-il pour sélectionner une parmi toutes les permutations de 10 cartes ?

Réponse :

1.2 Tri par sélection (selection sort)

En anglais :
to sort* = *trier.
a sorting algorithm* = *un algorithme de tri.

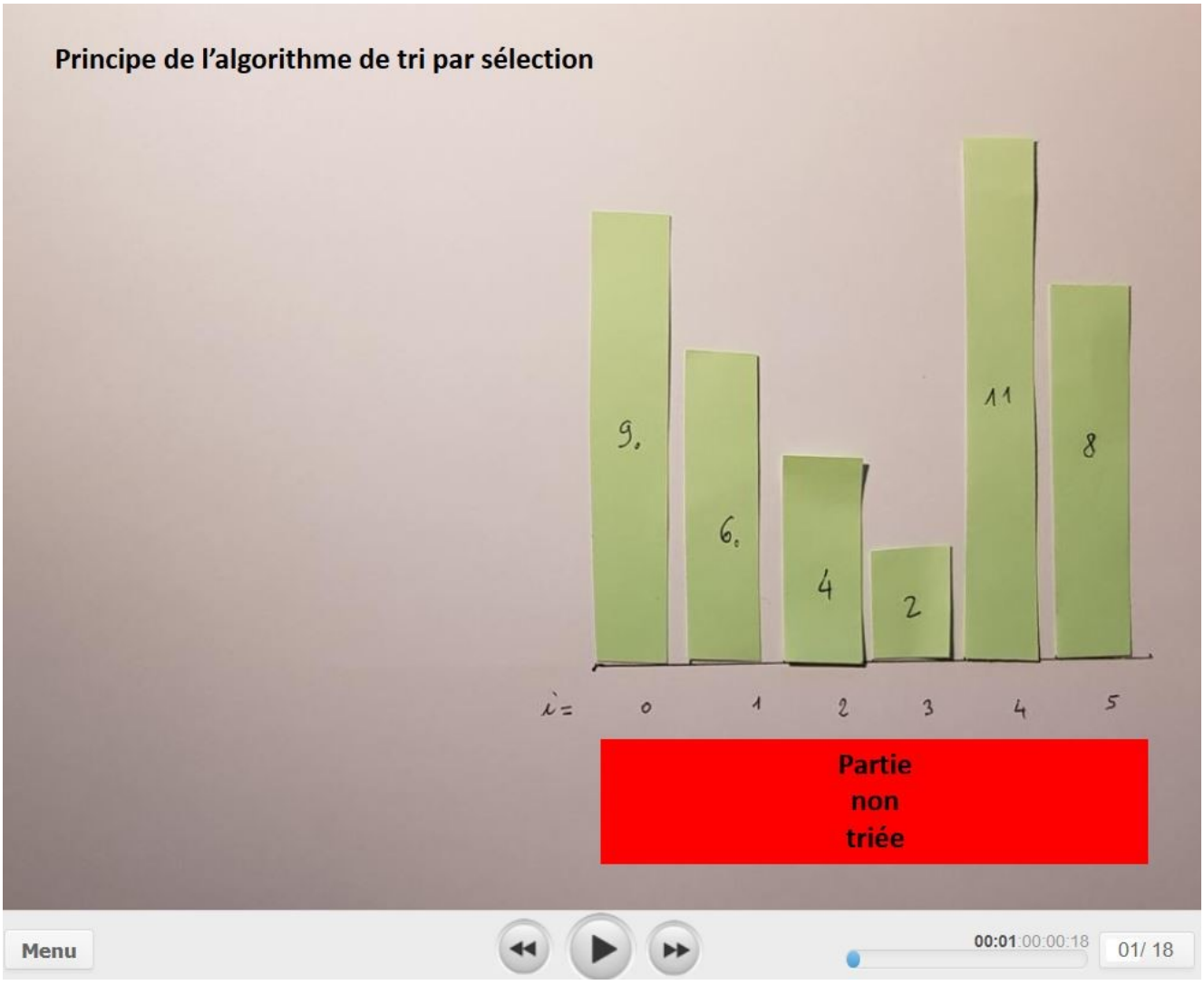
Sélection du plus petit

1.2.1 Le principe

Au départ, la partie non triée de la liste est égale à la liste entière.

1. Dans la partie non encore triée de la liste, on **sélectionne** le plus petit élément, c'est à dire qu'on repère son indice.
2. On *l'échange* avec le premier élément de la partie non encore triée.
3. Au début du tour suivant, la partie non triée a donc un élément de moins.

- Cliquez sur l'image ci-dessous et cliquez ensuite *sur la flèche plusieurs fois* pour faire évoluer le tri par sélection.



(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/tri_selection/tri_selection.html)

Lisez le paragraphe **Le principe** du milieu de la p. 310 au milieu de la p. 311

Voici l'algorithme du minimum :

- En entrée :
  - liste qui est une liste de n éléments depuis liste[0] jusqu'à liste[n-1]
  - i l'indice où commence la recherche du minimum.
- En sortie :
  - i_mini qui est l'indice du plus petit élément de la tranche liste[i : n]
  - liste[i_mini] qui est le plus petit élément de la tranche liste[i : n]

12) Complétez ci-dessous le code en Python de la fonction **mini** de façon à ce qu'elle corresponde à la docstring.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:

```
def mini(liste, i):

    """
    liste est une liste de n nombres à virgule flottante.
    La fonction donne l'indice du minimum et le minimum dans la tranche liste[i : len(l
iste)].

    Exemple : Si liste = [2.0, 35.0, 42.0, 39.0, 41.0] alors
    mini(liste, 2) renvoie l'indice du minimum et le minimum dans la tranche liste[2 :
5]
    c'est à dire i_mini = 3 et mini = 39.0

    Parametres nommes
    -----------------
    liste : de type list
    i : de type int
        L'indice du premier élément où commence la recherche du minimum.

    Retourne
    --------
    i_mini : de type int
             L'indice du plus petit des éléments de la tranche liste[i : len(liste)]
    liste[i_mini] : de type float
                    Le minimum trouvé dans la tranche liste[i : len(liste)]

    """
    #  Initialisation des variables de départ
    ## i_mini =
    ## mini =

    #  Parcours de la tranche liste[i+1 : len(liste)] et mise à jour de i_mini et mini.
    for j in range(i+1, len(liste)):
        if liste[j] < mini:
            ##
            ##
    return i_mini, mini
```

Testez votre fonction dans la cellule ci-dessous :

In [ ]:
```
liste = [2.0, 35.0, 42.0, 39.0, 41.0]
print(mini(liste, 2))
```

13) Complétez ci-dessous le code en Python de la fonction **echange_premier_mini** de façon à ce qu'elle corresponde à la docstring :

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
def echange_premier_mini(liste, i):

    """
    liste est une liste de n nombres à virgule flottante.
    La fonction échange les éléments liste[i] et liste[i_mini] dans la tranche liste[i
: len(liste)].
    Exemple : Si liste = [2.0, 35.0, 42.0, 39.0, 41.0] alors
    echange_premier_mini(liste, 2) examine la tranche [42.0, 39.0, 41.0]
    et échange le premier élément et le minimum.
    Après l'exécution on a liste = [2.0, 35.0, 39.0, 42.0, 41.0]

    Parametres nommes
    -----------------
    liste : de type list
    i : de type int

    Retourne
    --------
    Aucun retour car la liste est modifiée sur place.

    """

    #  Initialisation des variables de départ
    ## i_mini =
    ## mini =

    #  Parcours de la tranche liste[i+1 : len(liste)] et mise à jour de i_mini et mini.
    for j in range(i+1, len(liste)):
        if liste[j] < mini:
            ##
            ##
    ## liste[i], liste[i_mini] =
```

Testez votre fonction dans la cellule ci-dessous :

In [ ]:
```
ma_liste = [6222.0, 335.0, 421.0, 39.0, 431.0, 12.0]
echange_premier_mini(ma_liste, 2)
print(ma_liste)
```

Vous avez obtenu [6222.0, 335.0, 12.0, 39.0, 431.0, 421.0] comme attendu ? Si oui, vous pouvez continuer.

1.2.2 Programme en Python du tri par sélection

Lisez le paragraphe **Programme en Python** du milieu de la p. 311 au haut de la p. 312

- La fonction tri_selection consiste à répéter la fonction **echange_premier_mini(liste, i)**

dans une boucle for i allant de **0 à len(liste)-2**

```
0 est l'indice du premier élément de la liste.

len(liste)-1 est l'indice de l'avant dernier élément.
```

14) Complétez ci-dessous le tableau d'évolution de la liste :

Au départ, on a liste = [9, 6, 4, 2, 11, 8]

i	La fonction **echange_premier_mini(liste, i)** échange le 1er élément et le minimum sur la tranche non triée qui est ...	A la suite de quoi la liste vaut ...
0	liste[0:6] et qui vaut [9, 6, 4, 2, 11, 8]	[2, 6, 4, 9, 11, 8]
1	liste[1:6] et qui vaut [6, 4, 9, 11, 8]	[2, 4, 6, 9, 11, 8]
2	liste[2:6] et qui vaut [6, 9, 11, 8]	[2, 4, ..., ..., ..., ...]
3	liste[3:6] et qui vaut [9, 11, 8]	[2, 4, ..., ..., ..., ...]
4	liste[4:6] et qui vaut [11, 9]	[2, 4, ..., ..., ..., ...]

- La liste au départ était liste = [9, 6, 4, 2, 11, 8]

Après l'application de cinq tours de boucle **for i in range(0, 5)** on a obtenu la liste triée sur place, dans l'ordre croissant :

- La liste à l'arrivée est liste = [2, 4, 6, 8, 9, 11]

15) Complétez ci-dessous le code de la fonction tri_selection :

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
def tri_selection(liste):

    """
    liste est une liste de n nombres à virgule flottante.
    La fonction trie sur place, par ordre croissant, la liste donnée en argument.
    Exemple : Si liste = [9, 6, 4, 2, 11, 8] alors
    après l'exécution on a liste = [2, 4, 6, 8, 9, 11]

    Parametres nommes
    -----------------
    liste : de type list
            Une liste de nombres de type int

    Retourne
    --------
    Aucun retour car la liste est modifiée sur place.

    Remarque : Une fonction qui ne retourne rien est appelée "procédure".

    """


    for i in range(0, len(liste)-1):
        #  Initialisation des variables de départ
        ## i_mini =
        ## mini =

        #  Parcours de la tranche liste[i+1 : len(liste)] et mise à jour de i_mini et m
ini.
        for j in range(i+1, len(liste)):
            if liste[j] < mini:
                ##
                ##
        ## liste[i], liste[i_mini] =
```

16) Testez la fonction tri_selection avec une liste de votre choix :

In [ ]:
```
ma_liste = [..., ..., ..., ..., ...]
ma_liste_triee = list(ma_liste) #  Fait une copie de liste. Cela permet de conserver ma
_liste inchangée.
tri_selection(ma_liste_triee)  #  Fait le tri sur place de ma_liste_triee.

print(ma_liste)
print(ma_liste_triee)
```

1.2.3 Validité de l'algorithme du tri par sélection

Lisez le paragraphe **Validité de l'algorithme** du haut de la p. 312 au bas de la p. 312

- Preuve de terminaison

17) Justifiez la terminaison de l'algorithme du tri par sélection :

Réponse :

- Preuve de correction

On va prouver que l'invariant de boucle (la boucle for externe) :

```
- liste[0 : i+1] est déjà triée.
- liste[i+1 : n] ne contient que des éléments supérieurs à ceux de liste[0 : i+1].
```

est vrai pour chaque i pour i allant de 0 à len(liste)-2.

18) Complétez les pointillés dans la preuve de la correction de l'algorithme en utilisant l'invariant de boucle :

1) Initialisation

Réponse : Après le premier passage dans la boucle, pour i = 0, liste[0 : 1] ne contient.....

La propriété est donc vraie pour i = 0.

2) Hérédité

Réponse : Si on suppose qu'après le passage dans la boucle, pour i = k, liste[0 : k+1] est triée et tous les éléments de liste[k+1 : n] sont supérieurs à ceux de liste[0 : k+1], alors après le passage suivant, le minimum de liste[k+1 : n] est placé en position d'indice k+1. Ce minimum est supérieur à ..... et inférieur à .....

La propriété est donc vraie pour i = ...

3) Conclusion

La propriété (c'est à dire l'invariant de boucle) est vraie après le premier passage dans la boucle et elle est héréditaire.

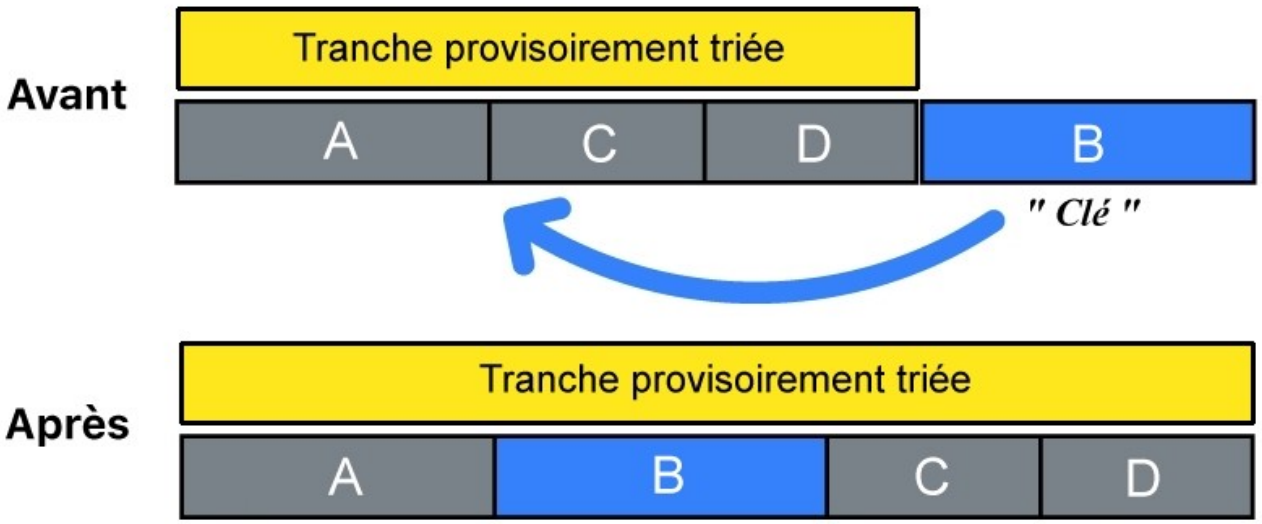
Donc elle est vraie pour tous les tours de boucle. En particulier elle est vraie après le dernier tour de boucle c'est à dire pour i = n-2.

A ce moment là :

- liste[0 : n-1] est .....
- L'élément d'indice n-1, le dernier de la liste, est .....

Donc liste[0 : n], soit toute la liste, est .....

1.3 Tri par insertion

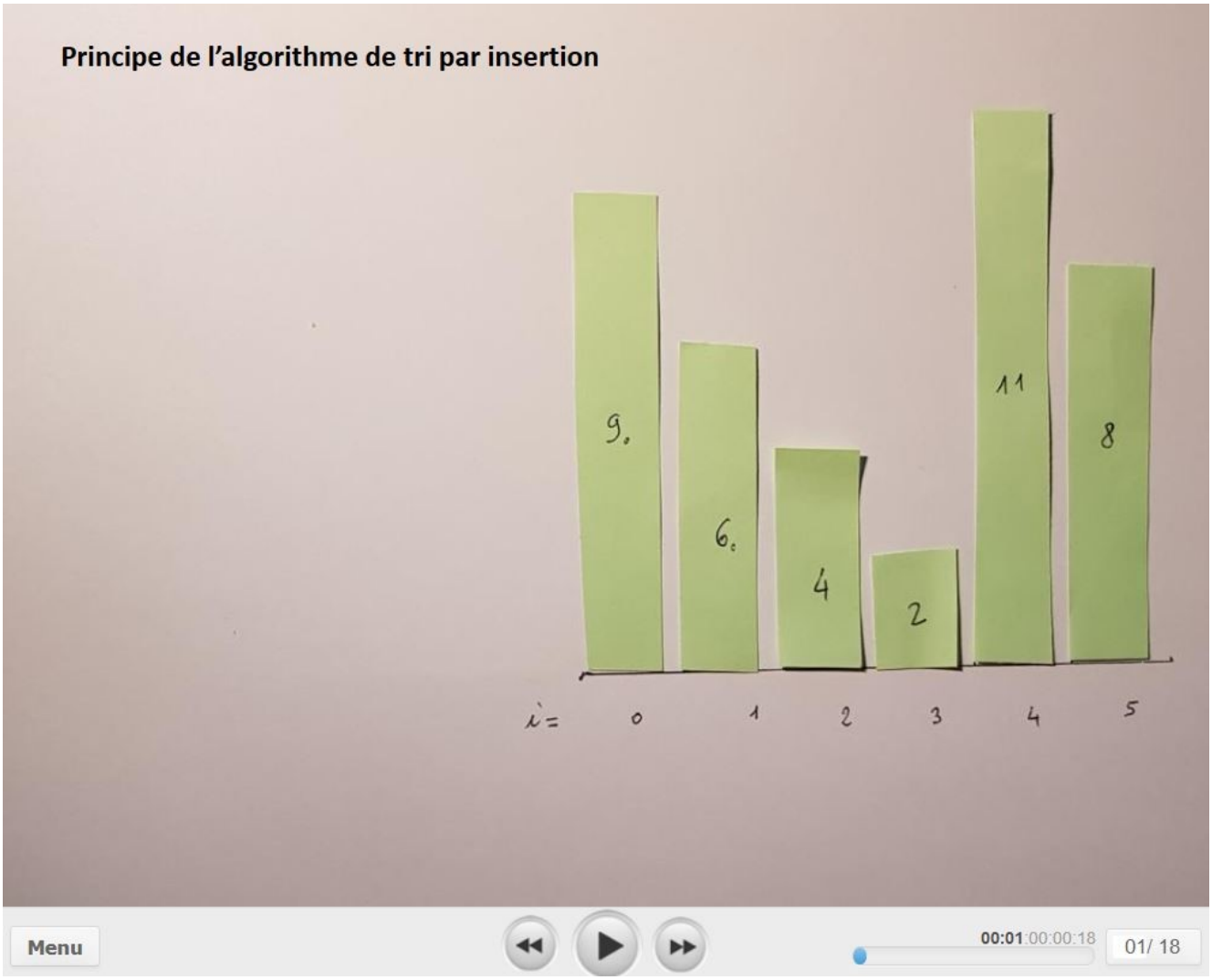


1.3.1 Le principe

Au départ, le 1er élément est la partie de la liste provisoirement triée. La partie non triée commence au deuxième élément.

1. Dans la partie non encore triée de la liste, on prend le premier élément. On l'appelle la "clé".
2. On l'**insère** à sa place dans la partie déjà triée.
3. Au début du tour suivant, la partie non triée a donc un élément de moins.

- Cliquez sur l'image ci-dessous pour démarrer le diaporama de présentation du **principe** du tri par insertion.



(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/tri_insertion/tri_insertion.html)

Lisez le paragraphe **Le principe** du haut de la p. 313 au bas de la p. 313

Voici l'algorithme d'insertion :

Il insère la clé liste[k] à sa place dans la liste partie de la liste provisoirement triée liste[0 : i+1]

Exemple : liste = [4, 6, 9, 2, 11, 8]

- La partie provisoirement triée est liste[0 : 3]
- L'indice de la clé est 3.
- La clé est liste[3] qui vaut 2.
- On insère la clé à sa place, ici au tout début en décalant certains éléments d'un rang à droite pour faire de la place.

Ainsi : liste = [2, 4, 6, 9, 11, 8]

- En entrée :
  - liste qui est une liste de n éléments depuis liste[0] jusqu'à liste[n-1]
  - i est l'indice du dernier élément de la partie provisoirement triée liste[0 : i+1]
- En sortie :
  - Aucune sortie : la liste est modifiée sur place.

19) Complétez ci-dessous le code en Python de la fonction **insertion** de façon à ce qu'elle corresponde à la docstring.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
def insertion(liste, i):

    """
    liste est une liste de n nombres à virgule flottante.
    La tranche liste[0: i+1] est provisoirement triée.
    i+1 est l'indice du premier élément de la tranche liste[i+1: len(liste)] non encore triée.
    liste[i+1] est la clé à insérer dans la tranche provisoirement triée.

    Ex : liste = [4.0, 6.0, 9.0, 2.0, 11.0, 8.0].
    Puisque la tranche liste[0 : 3] est déjà provisoirement triée, on va insérer la clé liste[3]
    qui vaut 2.0 à sa place par la commande insertion(liste, 2). Les éléments 4.0, 6.0, 9.0 sont
    décalés à droite pour faire de la place. La liste devient [2.0, 4.0, 6.0, 9.0, 11.0, 8.0].


    Parametres nommes
    -----------------
    liste : de type list
    i : de type int
        La liste est provisoirement triée de liste[0] à liste[i]

    Retourne
    --------
    Rien. La clé liste[i+1] est insérée sur place.

    """
    #  Calcule l'indice de la clé.
    ## k =
    #  Mémorise la clé.
    ## cle =

    #  Compare la clé liste[k] avec les éléments de tranche liste[0 : k].
    #  Tant que la clé est inférieure à l'élément liste[k - 1], celui-ci est décalé d'un cran à droite.
    while cle < liste[k-1] and k > 0:
        #  Décale d'un cran à droite liste[k - 1].
        ## liste[k] =
        #  Décrémente k d'une unité.
        ## k =
    #  Ecrit la clé à la bonne place dans la tranche de la liste provisoirement triée.
    ## liste[k] =
```

Testez votre fonction dans la cellule ci-dessous :

In [ ]:
```
ma_liste = [4.0, 6.0, 9.0, 2.0, 11.0, 8.0]
ma_liste_insertion = list(ma_liste)  # Copie de liste.
#  liste[0 : 3] est provisoirement triée, on insère la clé liste[3] qui vaut 2.0 à sa place.
insertion(ma_liste_insertion, 2)

print("ma_liste = ", ma_liste)
print("ma_liste_insertion = ", ma_liste_insertion)
```

1.3.2 Programme en Python du tri par insertion

Lisez le paragraphe **Programme en Python** du bas de la p. 313 au bas de la p. 314

- Soit une liste nommée liste[0 : n] qui contient n éléments de liste[0] à liste[n-1].
- La fonction tri_insertion consiste à répéter la fonction **insertion(liste, i)** de manière à augmenter progressivement la taille de la tranche provisoirement triée. Détaillons le fonctionnement :
  - i vaut 0.
    - La tranche liste[0 : 1] contient un seul élément. Elle est provisoirement triée.
    - La clé est l'élément suivant liste[1].
    - On l'insère à sa place dans la tranche liste[0 : 1] en faisant éventuellement un décalage à droite.
    - Donc liste[0 : 2] devient provisoirement triée.
  - i vaut 1.
    - La tranche liste[0 : 2] contient deux éléments. Elle est provisoirement triée.
    - La clé est l'élément suivant liste[2].
    - On l'insère à sa place dans la tranche liste[0 : 2] en faisant éventuellement des décalages à droite.
    - Donc liste[0 : 3] devient provisoirement triée.
  - etc...
  - i vaut n-1 qui est l'indice du dernier élément de liste[0 : n].
    - La tranche liste[0 : n-1] contient n-1 éléments. Elle est provisoirement triée.
    - La clé est l'élément suivant liste[n-1].
    - On l'insère à sa place dans la tranche liste[0 : n-1] en faisant éventuellement des décalages à droite.
    - Donc liste[0 : n] est toute la liste triée.

20) Complétez ci-dessous le tableau d'évolution de la liste :

Au départ, on a liste = [9, 6, 4, 2, 11, 8]

i	Indice de la clé k	clé	La fonction insertion(liste, i) insère la clé dans la tranche provisoirement triée...	A la suite de quoi la liste vaut ...	La tranche provisoirement triée vaut ...
0	1	6	liste[0:1] qui vaut [9]	[6, 9, 4, 2, 11, 8]	[6, 9]
1	2	4	liste[0:2] qui vaut [6, 9]	[4, 6, 9, 2, 11, 8]	[4, 6, 9]
2	3	2	liste[0:3] qui vaut [2, 4, 6]	[2, 4, 6, 9, ..., ...]	[2, 4, 6, 9]
3	4	11	liste[0:4] qui vaut [2, 4, 6, 9]	[2, 4, 6, 9, ..., ...]	[2, 4, 6, 9, 11]
4	5	8	liste[0:5] qui vaut [2, 4, 6, ..., ...]	[2, 4, 6, ..., ..., ...]	[2, 4, 6, 8, 9, 11]

- La liste au départ était liste = [9, 6, 4, 2, 11, 8]

Après l'application de cinq tours de boucle **for i in range(0, 5)** on a obtenu la liste triée sur place, dans l'ordre croissant :

- La liste à l'arrivée est liste = [2, 4, 6, 8, 9, 11]

21) Complétez ci-dessous le code de la fonction tri_insertion :

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:

```
def tri_insertion(liste):

    """
    liste est une liste de n nombres à virgule flottante.
    La fonction trie sur place, par ordre croissant, la liste donnée en argument.
    Exemple : Si liste = [9, 6, 4, 2, 11, 8] alors
    après l'exécution on a liste = [2, 4, 6, 8, 9, 11]

    Parametres nommes
    -----------------
    liste : de type list
            Une liste de nombres de type int

    Retourne
    --------
    Aucun retour car la liste est modifiée sur place.

    """

    for i in range(0, len(liste)-1):
        #  Calcule l'indice de la clé.
        ## k =
        #  Mémorise la clé.
        ## cle = liste[k]
        while cle < liste[k - 1] and k > 0:
            #  Décale d'un cran à droite liste[k - 1].
            ## liste[k] =
            #  Décrémente k d'une unité.
            ## k =
        #  Ecrit la clé à sa place dans la partie de a liste déjà triée.
        liste[k] =
```

22) Testez la fonction tri_selection avec une liste de votre choix :

In [ ]:
```
ma_liste2 = [..., ..., ..., ..., ...]
ma_liste_triee2 = list(ma_liste2) #  Fait une copie de liste. Cela permet de conserver
ma_liste inchangée.
tri_insertion(ma_liste_triee2)  #  Fait le tri sur place de ma_liste_triee.

print(ma_liste2)
print(ma_liste_triee2)
```

1.3.3 Validité de l'algorithme du tri par insertion

Lisez le paragraphe **Validité de l'algorithme** du bas de la p. 314 au milieu de p.315

- Preuve de terminaison

23) Justifiez la terminaison de l'algorithme du tri par insertion en complétant les pointillés :

Réponse : Il y a deux boucles imbriquées.

- La boucle externe a un nombre de tours déterminé et fini parce que c'est une boucle ...
- La boucle interne est une boucle while. Pour prouver qu'elle a un nombre fini de tours, on regarde un variant de boucle. Ce variant est ... dont les valeurs sont une suite d'entiers décroissante incluse elle-même dans la suite d'entiers décroissante allant de i + 1 à 1. Donc il y a au plus .....

- Preuve de correction

On va prouver que l'invariant de boucle (la boucle for externe) :

```
a) liste est une permutation de la liste de départ (c'est à dire composée d'exactement les
mêmes éléments, éventuellement placés dans un ordre différent).

b) liste[0:i+2] est triée (voir à la question 20 la dernière colonne du tableau).
```

est vrai pour chaque i pour i allant de 0 à len(liste)-2.

24) Complétez les pointillés dans la preuve de la correction de l'algorithme en utilisant l'invariant de boucle :

```
a) A chaque tour de boucle for, on insère la clé à sa place dans la partie provisoirement
triée, en décalant certains éléments. Cela prouve qu'à chaque tour de boucle, la liste est
une permutation des éléments de la liste de départ.
```

```
b) Montrons que pour tout i on a "liste[0:i+2] est triée".
```

1) Initialisation

Réponse : Après le premier passage dans la boucle, pour i = 0, liste[0:2] est .....(voir le tableau de la question 20) à la dernière colonne de la première ligne).

La propriété est donc vraie pour i = 0.

2) Hérédité

Réponse : Si on suppose qu'après le passage dans la boucle, pour i = k, liste[0 : k+2] est triée. Alors, au passage suivant, l'élément liste[k+2] est inséré à sa place parmi les éléments de la tranche liste[0 : k+2] en décalant éventuellement vers la droite certains éléments, ce qui forme la tranche liste[0 : k+3] qui est triée.

La propriété est donc vraie pour i égal à .....

3) Conclusion

La propriété (c'est à dire l'invariant de boucle) est vraie après le premier passage dans la boucle et elle est héréditaire.

Donc elle est vraie pour tous les tours de boucle. En particulier elle est vraie après le dernier tour de boucle c'est à dire pour i = n-2.

A ce moment là :

- La liste liste[0 : n-2+2] est .....

Donc la liste liste[0 : n], soit toute la liste, est .....

25) De quel ordre est le coût, en nombre de comparaisons, dans le pire des cas (la liste est complètement inversée) et dans le cas moyen (une liste de nombres au hasard), d'un tri par insertion d'une liste de longueur n ?

Réponse :

26) De quel ordre est le coût, en nombre de comparaisons, pour une liste presque triée (la liste était déjà triée et on vient y ajouter quelques éléments), d'un tri par insertion d'une liste de longueur n ?

Réponse :

27) Lorsqu'on veut trier une liste presque déjà triée, quelle est la méthode la moins coûteuse pour effectuer le tri ?

Réponse :

Visionnez la vidéo ci-dessous qui présente trois tris basiques

- Le tri par sélection
- Le tri à bulle (qui n'est pas au programme - c'est de la culture générale)
- Le tri par insertion

puis répondez aux deux questions qui suivent :

(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/tris_basiques.mp4)

28) En quelle année a été utilisé pour la première fois un tri mécanique automatisé ?

Réponse :

29) Quels sont les deux sortes de tris non basiques cités dans cette vidéo ?

Réponse :

30) Dans la vidéo, à 2:51, il est dit "le tri par sélection est un des algorithmes de tri parmi les plus *triviaux*". Quelle est la signification du mot *trivial* ici ?

Réponse :

1.4 Application à la médiane et aux quantiles

Lisez le paragraphe **Application** dans le bas de la p. 315

31) Lorsqu'on calcule la médiane d'une liste, il faut d'abord qu'elle soit triée. Cette précondition étant vérifiée, quelles formules permettent de calculer la médiane d'une liste triée de longueur n ?

Réponse :

- Si la longueur n de la liste est impaire alors Médiane = ...
  Ex : liste = [elt0, elt1, elt2, elt3, elt4]
  Médiane = liste[(5 - 1)//2]
  Médiane = liste[2]
  Médiane = ...

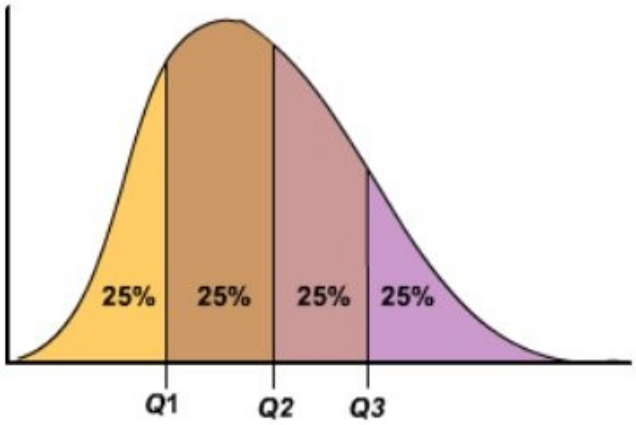
- Si la longueur n de la liste est paire alors Médiane = ...
  Ex : liste = [elt0, elt1, elt2, elt3, elt4, elt5]
  Médiane = (liste[(6 - 2)//2] + liste[6//2]) / 2
  Médiane = (liste[2] + liste[3]) / 2
  Médiane = ...

Quantiles

- On a des définitions et des algorithmes similaires pour trouver les valeurs qui séparent la série de valeurs triées en 4 parts ayant chacune 25% des effectifs ou bien en 10 parts ayant chacune 10% des effectifs ou bien en 100 parts ayant chacune 1% des effctifs etc. Ces valeurs sont appelées dans le cas général des **quantiles**. Illustrons avec les cas les plus courants :

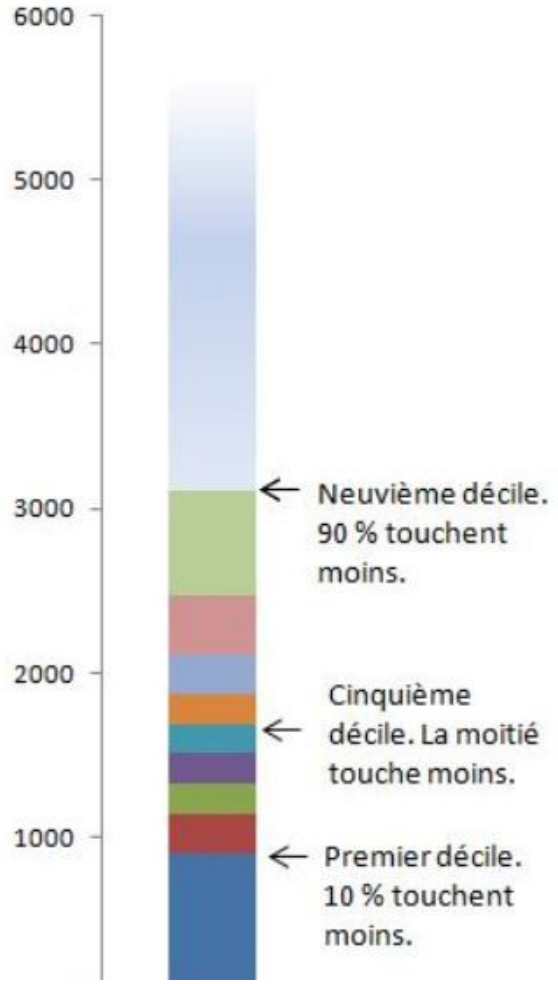
Quartiles

- Exemple : les 1er, 2e et 3e quartiles :



Déciles

- Exemple : les 1er, 2e ... 9e déciles :

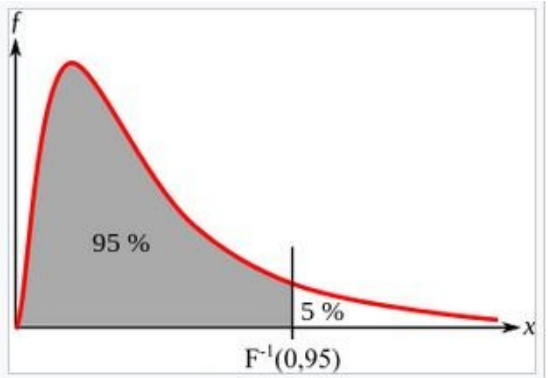


Revenus mensuels

Centiles

Un centile est chacune des 99 valeurs qui divisent les données triées en 100 parts égales, de sorte que chaque partie représente 1/100 de l'échantillon de population.

- Exemple : le 95e centile :



Définition du 95e centile d'une loi

1.5 Tri avec la fonction sorted ou la méthode .sort()

Lisez le paragraphe **Le tri en Python** du bas de la p. 315 au haut de la p. 316

32) Lorsqu'on veut trier une liste, on peut utiliser la fonction ma_liste_triee = sorted(ma_liste). Dans la cellule ci-dessous, saisissez une liste de votre choix d'effectif impair.

In [ ]:
```
ma_liste = [..., ..., ..., ..., ...]
ma_liste_triee = sorted(ma_liste)

print(ma_liste)
print(ma_liste_triee)
```

33) Recopiez la liste ma_liste de la question 32 et complétez le code pour calculer la médiane de ma_liste dans la cellule ci-dessous.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
# Exemple de liste à effectif impair.
ma_liste = [..., ..., ..., ..., ...]
ma_liste_triee = sorted(ma_liste)

## n =
## Mediane =
print(Mediane)
```

34) Lorsqu'on veut trier une liste, on peut aussi utiliser la méthode ma_liste2.sort(). Dans la cellule ci-dessous, saisissez une liste de votre choix d'effectif pair et appliquez-lui cette méthode .sort()

In [ ]:
```
ma_liste2 = [..., ..., ..., ..., ..., ...]
ma_liste2.sort()

print(ma_liste2)
```

35) Recopiez la liste ma_liste2 de la question 33 puis calculez la médiane de ma_liste2 dans la cellule ci-dessous.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
#  Exemple de liste à effectif pair.
ma_liste2 = [..., ..., ..., ..., ..., ...]
ma_liste2.sort()

## n =
## Mediane =
print(Mediane)
```

36) Quel est le nom de l'algorithme de tri utilisé dans la fonction sorted intégrée dans Python ?

Réponse :

Paramètre reverse dans la fonction sorted(liste, reverse=True)

37) On donne une liste. Observez les résultats du tri par la fonction sorted selon qu'on précise ou non la valeur du paramètre reverse=True

In [ ]:
```
ma_liste = ['Sibylle', 'Margaux', 'Paul', 'Apolline']

ma_liste_triee = sorted(ma_liste)
ma_liste_triee_reverse = sorted(ma_liste, reverse=True)

print('ma_liste = ', ma_liste)
print('ma_liste_triee = ', ma_liste_triee)
print('ma_liste_triee_reverse = ', ma_liste_triee_reverse)
```

Tri d'un tableau selon une colonne particulière avec la fonction sorted

38) On donne un tableau où ont été enregistrées les notes de quatre élèves. Par convention, la première est celle de math, la deuxième est celle de français. Observez le code Python puis les résultats du tri par la fonction sorted selon la valeur du paramètre key.

Nom	Mathématiques	Français
Sibylle	12,5	13,8
Margaux	11,7	6,9
Paul	4,4	8,9
Apolline	18,6	20,0

In [ ]:
```
mon_tableau = [['Sibylle', 12.5, 13.8], ['Margaux', 11.7, 6.9], ['Paul', 4.4, 8.9], ['Apolline', 18.6, 20.0]]


#  On commence par définir des fonctions qui peuvent être appelées
#  sur chaque enregistrement en tant que 'key' pour le tri.

def nom(enregistrement):
    """
    Renvoie le nom présent dans l'enregistrement (un "enregistrement" correspond à une ligne du tableau
    c'est à dire, ici, à une liste représentant un élève).

    Parametres nommes
    -----------------
    enregistrement : de type list
                     Contient le nom, la note de math, la note de français

    Retourne
    --------
    Le premier élément de la liste.

    """

    return enregistrement[0]



def math(enregistrement):
    """
    Renvoie le nom présent dans l'enregistrement

    Parametres nommes
    -----------------
    enregistrement : de type list
                     Contient le nom, la note de math, la note de français

    Retourne
    --------
    Le deuxième élément de la liste.

    """

    return enregistrement[1]



def francais(enregistrement):
    """
    Renvoie le nom présent dans l'enregistrement

    Parametres nommes
    -----------------
    enregistrement : de type list
                     Contient le nom, la note de math, la note de français

    Retourne
    --------
    Le troisième élément de la liste.
```

```
    """
    return enregistrement[2]


mon_tableau_selon_nom = sorted(mon_tableau, key=nom)
mon_tableau_selon_francais = sorted(mon_tableau, key=francais)
mon_tableau_selon_math = sorted(mon_tableau, key=math)

print('mon_tableau = ', mon_tableau)

print('mon_tableau_selon_nom = ', mon_tableau_selon_nom)
print('mon_tableau_selon_math = ', mon_tableau_selon_math)
print('mon_tableau_selon_francais = ', mon_tableau_selon_francais)
```

39) Il est possible de préciser dans la fonction sorted *à la fois les deux paramètres* key et reverse. Complétez le code ci-dessous pour afficher le tableau trié selon les notes de math décroissantes.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation de variable etc). Vous supprimez ensuite les deux ##.
```

Nom	Mathématiques	Français
Sibylle	12,5	13,8
Margaux	11,7	6,9
Paul	4,4	8,9
Apolline	18,6	20,0

In [ ]:
```
mon_tableau = [['Sibylle', 12.5, 13.8], ['Margaux', 11.7, 6.9], ['Paul', 4.4, 8.9], ['Apolline', 18.6, 20.0]]

def nom(enregistrement):
    #  Retourne le Nom d'un enregistrement (c'est à dire d'une liste).
    return enregistrement[0]



def math(enregistrement):
    #  Retourne la note de Math d'un enregistrement.
    return enregistrement[1]



def francais(enregistrement):
    #  Retourne la note de Français d'un enregistrement.
    return enregistrement[2]

mon_tableau_selon_math_decroissant = sorted(mon_tableau, key=..., reverse=...)
print('mon_tableau_selon_math_decroissant = ', mon_tableau_selon_math_decroissant)
```

2. Les algorithmes gloutons

2.1 Introduction

Lisez le paragraphe **Algorithmes gloutons** au milieu de la p. 322

40) Dans un problème d'optimisation une des deux caractéristiques est une fonction, le but étant de trouver les valeurs des variables de façon à rendre maximale (ou minimale) cette fonction. Mais de quelle autre caractéristique doit-on tenir compte ?

Réponse :

41) On pourrait, pour trouver la solution optimale, essayer toutes les combinaisons possibles des valeurs de toutes les variables. Mais si on procède de cette façon, l'algorithme est souvent inutilisable. Pourquoi ?

Réponse :

42) Un algorithme du type "glouton" (nous allons étudier ce type après) est assez simple. Mais quel inconvénient présente-t-il ?

Réponse :

illustration de la problématique : Une série de choix localement optimaux vs. une solution optimale globale.

- Sur la figure suivante, on cherche à optimiser la fonction f qui donne l'altitude. En l'occurrence, on cherche un chemin qui permet d'obtenir un maximum de f.

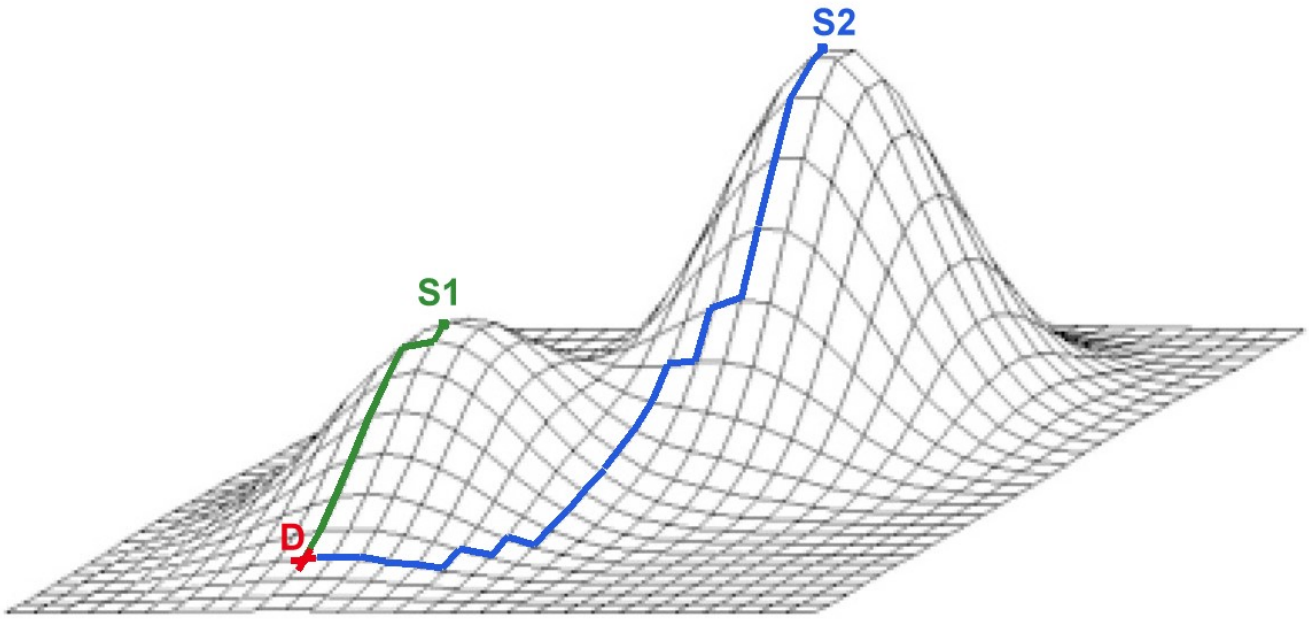
En partant du point **D**, on peut avoir deux stratégies :

1) Soit avancer, à chaque pas, dans la direction qui fait gagner le plus d'altitude. C'est **l'algorithme glouton** qui consiste à faire une série de choix *localement optimaux*. L'algorithme glouton, dans ce cas, arrive à une solution S1 qui est un maximum local.

2) Soit étudier tous les chemins possibles en partant de D et comparer toutes les altitudes atteintes au cours des chemins. Au final, on retient un chemin qui atteint la solution optimale globale S2 (il n'y a pas plus haut que S2). Mais, comme vu à la question 41, **l'algorithme d'énumération toutes les possibilités de façon exhaustive** est souvent inutilisable car trop couteuse.

- Remarque :

Ici, l'algorithme glouton ne donne pas la solution optimale S2. Mais, dans d'autres conditions, par exemple si D était situé au pied de la plus grande montagne, alors l'algorithme glouton donnerait la solution optimale.



On retient :

- Les algorithmes gloutons ne sont pas trop coûteux, donc sont exécutables en un temps raisonnable.
- La solution qu'ils donnent est bonne.
- Mais on n'est pas certain que ce soit la solution optimale.

2.2 Problème du sac à dos

Lisez le paragraphe **Le problème du sac** à dos du bas de la p. 322 au bas de la p. 324

Un voleur est en train de cambrioler une maison : il repère 6 objets, ayant tous une certaine valeur et un certain poids. Comment va-t-il choisir les objets à emporter sachant qu'il ne peut pas mettre dans son sac à dos plus de 15 Kg ?

Objet	Valeur	Poids	Valeur/Poids
Objet 1	126	14	9
Objet 2	32	2	16
Objet 3	20	5	4
Objet 4	5	1	5
Objet 5	18	6	3
Objet 6	80	8	10

Il y a plusieurs types de choix possibles.

- Le voleur préfère emporter d'abord les objets qui ont la plus grande valeur (en €).
- Le voleur préfère emporter d'abord les objets qui ont le plus petit poids (en Kg) ou - ce qui revient au même - l'inverse du poids le plus grand.
- Le voleur préfère emporter d'abord les objets qui ont le plus grand rapport Valeur/Poids (en €/Kg).

On appelle cela des heuristiques.

"**Une heuristique est un raisonnement** formalisé de résolution de problème dont on tient pour plausible, mais non pour certain, qu'il conduira à la détermination d'une solution satisfaisante du problème."

- Pour savoir quelle est la meilleure heuristique, on programme un algorithme glouton.
  - Si la première heuristique est suivie, l'algorithme glouton va trier le tableau par Valeurs décroissantes puis il prendra (ou non) dans cet ordre les objets tant que le cumul des poids reste inférieur ou égal à 15.
  - Si la deuxième heuristique est suivie, l'algorithme glouton va trier le tableau par inverses des Poids décroissants puis il prendra (ou non) dans cet ordre les objets tant que le cumul des poids reste inférieur ou égal à 15.
  - Si la troisième heuristique est suivie, l'algorithme glouton va trier le tableau par rapports Valeurs/Poids décroissants puis il prendra (ou non) dans cet ordre les objets tant que le cumul des poids reste inférieur ou égal à 15.
  - L'objet examiné est pris (ou non) si le cumul des poids avec ce nouvel objet reste inférieur ou égal à 15 Kg (ou non).

Nous comparerons ensuite la valeur totale du contenu du sac à dos selon les trois critères "Valeur", "inverse de Poids", "Valeur/Poids"

43) Complétez les codes suivants et exécutez-les :

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

1. Création des fonctions qui pourront être utilisées par le paramètre key pour trier la tableau selon le type d'heuristique choisie.

In [ ]:

```
mon_tableau = [['Objet 1', 126, 14], ['Objet 2', 32, 2], ['Objet 3', 20, 5],\
               ['Objet 4', 5, 1], ['Objet 5', 18, 6], ['Objet 6', 80, 8]]

def valeur(enregistrement):
    #  Retourne la Valeur d'un enregistrement (c'est à dire d'une liste).
    return  enregistrement[1]



def inverse_poids(enregistrement):
    #  Retourne l'inverse du Poids.
    ## return



def rapport(enregistrement):
    #  Retourne le quotient Valeur/Poids.
    ## return


## mon_tableau_selon_valeur_decroissante =
print('mon_tableau_selon_valeur_decroissante = ', mon_tableau_selon_valeur_decroissant
e)
```

1. Création de la fonction glouton

In [ ]:
```
def glouton(tableau, poids_max, heuristique):
    """
    Retourne la liste des objets à mettre dans le sac à dos et la valeur du sac


    Parametres nommes
    -----------------
    tableau : de type liste de listes
              Contient autant d'enregistrements [Nom de l'objet, Valeur, Poids, Valeur/
Poids]
              que d'objets à voler dans la maison.

    poids_max : de type int
                Le poids à ne pas dépasser.

    heuristique : de type fonction
                  Une fonction qui retourne une des données de l'enregistement.


    Retourne
    --------
    reponse , valeur : de type (list, int)
                       reponse est la liste les objets à mettre dans le sac.
                       valeur est la valeur totale des objets dans le sac.

    """

    tableau_trie = sorted(tableau, key=heuristique, reverse=True)  # Crée une copie du
tableau trié selon une
                                                                   # des heuristiques par or
dre décroissant.
    reponse = []  #  Initialisation de la liste d'objets à mettre dans le sac à dos.
    valeur = 0
    poids = 0

    i = 0  #  Initialisation du nombre de tours de boucles while.
    while i < len(tableau) and poids <= poids_max:
        nom, val, pds = tableau_trie[i]  #  Lecture des informations présentes sur l'en
registrement d'indice i.
        if poids + pds <= poids_max:  # Teste si, en ajoutant l'objet suivant sur la li
ste, cela respecte la contrainte.
            reponse.append(nom)  #  Ajoute le nom de l'objet dans la liste réponse.
            poids = poids + pds  #  Met à jour le poids du sac à dos.
            valeur = valeur + val  #  Met à jour la valeur du sac à dos.
        i = i + 1
    return reponse, valeur
```

1. Exécution de la fonction glouton selon les trois heuristiques possibles :

1) heuristique = valeur

In [ ]:
```
mon_tableau = [['Objet 1', 126, 14], ['Objet 2', 32, 2], ['Objet 3', 20, 5],\
               ['Objet 4', 5, 1], ['Objet 5', 18, 6], ['Objet 6', 80, 8]]

liste_sac, la_valeur = glouton(mon_tableau, 15, valeur)  #  Le tri du tableau se fait s
elon la valeur de
                                                         #  key renvoyée par la fonctio
n 'valeur'
print("Le sac = ", liste_sac)
print("La valeur = ", la_valeur)
```

- Explication : L'algorithme glouton trie le tableau avec *key=valeur* et *reverse=True* (ce qui revient à trier par Valeurs décroissantes).

A prendre	Objet	Valeur	Poids	Valeur/Poids
√	Objet 1	126	14	9
X	Objet 6	80	8	10
X	Objet 2	32	2	16
X	Objet 3	20	5	4
X	Objet 5	18	6	3
√	Objet 4	5	1	5

- L'algorithme passe en revue les enregistrements dans le tableau trié à partir de la première ligne, et prend dans le sac à dos les objets à condition que le poids cumulé reste inférieur ou égal à poids max = 15 Kg :
  - Le poids de l'objet en première ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 14 Kg.
  - Le poids cumulé + poids de l'objet en deuxième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en troisième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en quatrième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en cinquième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en sixième ligne est inférieur ou égal à 15 Kg donc il n'est pas ajouté à la liste reponse.
    - Le poids cumulé = 15 Kg.
- La valeur des objets du sac à dos est 126 + 5 = 131.

2) heuristique = inverse_poids

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
mon_tableau = [['Objet 1', 126, 14], ['Objet 2', 32, 2], ['Objet 3', 20, 5],\
               ['Objet 4', 5, 1], ['Objet 5', 18, 6], ['Objet 6', 80, 8]]


#  Le tri du tableau se fait selon la valeur de key renvoyée par la fonction 'inverse_p
oids'
## liste_sac, la_valeur =
print("Le sac = ", liste_sac)
print("La valeur = ", la_valeur)
```

- Explication : L'algorithme glouton trie le tableau avec *key=inverse_poids* et *reverse=True* (ce qui revient à trier par poids croissants).

A prendre	Objet	Valeur	Poids	Valeur/Poids
√	Objet 4	5	1	5
√	Objet 2	32	2	16
√	Objet 3	20	5	4
√	Objet 5	18	6	3
X	Objet 6	80	8	10
X	Objet 1	126	14	9

- L'algorithme passe en revue les enregistrements dans le tableau trié à partir de la première ligne, et prend dans le sac à dos les objets à condition que le poids cumulé reste inférieur ou égal à poids max = 15 Kg :
  - Le poids de l'objet en première ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 1 Kg.
  - Le poids cumulé + poids de l'objet en deuxième ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 3 Kg.
  - Le poids cumulé + poids de l'objet en troisième ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 8 Kg.
  - Le poids cumulé + poids de l'objet en quatrième ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 14 Kg.
  - Le poids cumulé + poids de l'objet en cinquième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en sixième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
- La valeur des objets du sac à dos est 5 + 32 + 20 + 18 = 75.

3) heuristique = rapport

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
mon_tableau = [['Objet 1', 126, 14], ['Objet 2', 32, 2], ['Objet 3', 20, 5],\
               ['Objet 4', 5, 1], ['Objet 5', 18, 6], ['Objet 6', 80, 8]]


#  Le tri du tableau se fait selon la valeur de key renvoyée par la fonction 'rapport'
## liste_sac, la_valeur =
print("Le sac = ", liste_sac)
print("La valeur = ", la_valeur)
```

- Explication : L'algorithme glouton trie le tableau avec *key=rapport* et *reverse=True* (ce qui revient à trier par Valeur/Poids décroissants).

A prendre	Objet	Valeur	Poids	Valeur/Poids
√	Objet 2	32	2	16
√	Objet 6	80	8	10
X	Objet 1	126	14	9
√	Objet 4	5	1	5
X	Objet 3	20	5	4
X	Objet 5	18	6	3

- L'algorithme passe en revue les enregistrements dans le tableau trié à partir de la première ligne, et prend dans le sac à dos les objets à condition que le poids cumulé reste inférieur ou égal à poids max = 15 Kg :
  - Le poids de l'objet en première ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 2 Kg.
  - Le poids cumulé + poids de l'objet en deuxième ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 10 Kg.
  - Le poids cumulé + poids de l'objet en troisième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
    - Le poids cumulé = 10 Kg.
  - Le poids cumulé + poids de l'objet en quatrième ligne est inférieur ou égal à 15 Kg donc il est ajouté à la liste reponse.
    - Le poids cumulé = 11 Kg.
  - Le poids cumulé + poids de l'objet en cinquième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
  - Le poids cumulé + poids de l'objet en sixième ligne est supérieur à 15 Kg donc il n'est pas ajouté à la liste reponse.
- La valeur des objets du sac à dos est 32 + 80 + 5 = 117.

Conclusion :

- Selon l'heuristique choisie, la valeur du sac à dos est 131 ou 75 ou 117. Le voleur choisira donc la première heuristique, celle qui consiste à choisir les objets dans l'ordre des valeurs décroissantes.
- 131 est une bonne solution, mais on n'est pas certain qu'elle soit optimale.
- Par une méthode exhaustive (c'est à dire une étude complète de toutes les façons de choisir des objets parmi les 6 objets tout en respectant la contrainte sur le poids cumulé), on montre que le choix optimal est :

A prendre	Objet	Valeur	Poids	Valeur/Poids
X	Objet 1	126	14	9
√	Objet 2	32	2	16
√	Objet 3	20	5	4
X	Objet 4	5	1	5
X	Objet 5	18	6	3
√	Objet 6	80	8	10

- La poids cumulé est 2 + 5 + 8 = 15 donc la contrainte poids cumulé inférieur ou égal à 15 Kg est respectée.
- La valeur des objets du sac à dos est 32 + 20 + 80 = 132 qui est la solution optimale.
- Avec peu d'objets comme ici, la méthode exhaustive est encore possible. Mais avec un grand nombre d'objets elle est beaucoup trop coûteuse en temps de calcul.

On retient :

- Algorithme glouton : construit une solution de manière incrémentale (c'est à dire pas à pas), en optimisant un critère de manière locale.

Nous allons voir encore deux problèmes que nous allons résoudre par un algorithme glouton.

2.3 Problème du rendu de monnaie

Lisez le paragraphe **Problème du rendu de monnaie** du bas de la p. 324 au milieu de la p. 326

44) Soit le système de pièces de la monnaie euros p = (1, 2, 5, 10, 20, 50, 100, 200) et r la somme en centimes qu'il faut rendre. On considère *la liste des nombres de pièces de chaque sorte* $[x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]$. Par exemple, il y a x_3 pièces de 10 centimes dans le rendu de monnaie. Quelle fonction cherche-t-on à rendre minimale ?

Réponse :

45) Soit le système de pièces de la monnaie euros p = (1, 2, 5, 10) et on suppose que $r = 12$ la somme en centimes qu'il faut rendre. Ecrivez un programme qui donne de façon exhaustive toutes les listes possibles des nombres de pièces de 1, 2, 5 et 10 centimes $[x_0, x_1, x_2, x_3]$ qu'on peut trouver pour composer cette somme rendue.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
p = (1, 2, 5, 10)
# Il peut y avoir de 0 à 12 pièces de 1 centime dans le rendu de monnaie (puisqu'il fau
t rendre r = 12 centimes).
for i in range(13):
    # Il peut y avoir de 0 à 6 pièces de 2 centimes dans le rendu de monnaie (puisqu'il
faut rendre r = 12 centimes).
    for j in range(7):
        # Il peut y avoir de 0 à 2 pièces de 5 centimes dans le rendu de monnaie (puisq
u'il faut rendre r = 12 centimes).
        ## for k in ...
            # Il peut y avoir de 0 ou 1 pièce de 10 centimes dans le rendu de monnaie
(puisqu'il faut rendre r = 12 centimes).
            ## for .......
                ## s =
                if s == 12:
                    print("Un rendu de monnaie possible en nombre de pièces de 1, 2, 5,
10 est :")
                    print([i, j, k, l])
```

46) Parmi toutes les listes obtenues, quelle est celle qui minimise le nombre de pièce ?

Réponse :

47) Complétez ci-dessous la fonction glouton. On remarque cet algorithme glouton prend deux paramètres et non trois comme l'algorithme du sac à dos.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
def glouton(p, r):
    """
    Retourne la liste des nombres de pièces de chaque sorte pour obtenir la somme r.


    Parametres nommes
    -----------------
    p : de type tuple
        Le tuple des valeurs des pièces dans un système monétaire donné.

    r : de type int
        La somme à rendre.

    Retourne
    --------
    solution : de type list
               La liste des nombres de pièces de chaque sorte

    """

    #  n est le nombre de sortes de pièces.
    ## n =
    #  On commence par la pièce de plus forte valeur
    i = n - 1
    #  On initialise une liste comportant n fois le nombre 0.
    ## solution =
    while r > 0:
        # On cherche la valeur de la plus grosse pièce qui est inférieure ou égale à r.
        while p[i] > r:
            i = i - 1
        # On incrémente le nombre de pièces qui ont pour valeur p[i].
        ## solution[i] =
        # On calcule le rendu de monnaie qu'il reste encore à rendre.
        ## r =
    return solution
```

- Vérifiez en exécutatnt le code ci-dessous que votre algorithme donne un nombre de pièces minimal pour rendre la monnaie quand la somme à rendre est r = 12 centimes.

In [ ]:
```
p = (1, 2, 5, 10, 20, 50, 100, 200)  # On utilise le système de pièces de la monnaie Euro.
glouton(p, 12)
```

L'algorithme glouton du rendu de monnaie donne-t-il toujours la solution optimale (c'est à dire le rendu avec le moins de pièces possible) ?

- On démontre que l'algorithme glouton donne toujours le rendu de monnaie optimal avec le système de pièces (1, 2, 5, 10, 20, 50, 100, 200) tel que celui des pièces d'euros. On dit que c'est un système de pièces "canonique".
- Cependant, il ne donne pas toujours le rendu de monnaie optimal avec un système non canonique. Un exemple de système non canonique : des pièces de 1, 3 et 4. Par exemple pour rendre r = 6, l'algorithme glouton donne comme solution 4 (la plus grosse pièce inférieure ou égale à 6) + 1 + 1. Il rend 3 pièces donc. On voit qu'il n'a pas trouvé la solution optimale qui est 3 + 3 (2 pièces seulement).
- Un tel système de pièces non canonique avait cours au Royaume-Uni avant 1971.

2.4 Problème des stations d'essence

Lisez le paragraphe **Problème des stations d'essence** du milieu de la p. 326 à la p. 328

- Exécutez le code suivant qui donne un exemple d'affichage d'un parcours entre un point de départ **D** et un point d'arrivée **A**. Il y a 7 stations intermédiaires nommées **S0, S1, ... , S6**. On a les 8 distances suivantes :
  - d0 distance de A à S1 : 137 km
  - d1 distance de S1 à S2 : 96 km
  - d2 distance de S2 à S3 : 105 km
  - d3 distance de S3 à S4 : 128 km
  - d4 distance de S4 à S5 : 132 km
  - d5 distance de S5 à S6 : 88 km
  - d6 distance de S6 à A : 76 km
  - d7 distance de S3 à S4 : 51 km

In [ ]:
```
#  Importation des bibliotheques permettant les graphiques
%matplotlib inline
import matplotlib.pyplot as plt   # Importation pour dessiner.
import math  #  Importation pour avoir les fonctions cos et sin.
import random  #  Importation pour avoir un angle au hasard

#  Definition de la liste des distances


# Création de la fonction translation du point précédente au suivant
def t(r, p):
    """
    Translate le point p d'un vecteur de norme r et de direction aléatoire

    Paramètres nommés
    -----------------
        r : de type float
            La longueur du vecteur de la translation.
        p : de type tuple de nombres de type float
            Les coordonnées (x, y) du point.

     Retourne
     --------
         x, y : de type tuple de nombres de type float
                Le coordonnées du point image par la translation.

    """
    theta = float(random.randint(0, 90))
    theta = theta * math.pi/180  # Conversion de l'angle aléatoire en radians.
    x = p[0] + r * math.cos(theta)
    y = p[1] + r * math.sin(theta)
    return x, y



def affiche_figure(liste_distances):
    #  Construction de la liste des noms, des abscisses, des ordonnées.
    liste_noms = ['D']  #  Le premier point est D (Départ).
    liste_x = [0]  #  Initialisation avec l'abscisse de D.
    liste_y = [0]  #  Initialisation avec l'ordonnée de D.

    #  On construit le point suivant en fonction du précédent, en tenant compte de leur
distance.
    for i in range(len(liste_distances)):
        mon_nom = 'S' + str(i)  #  Création du nom du nouveau point.

        ma_norme = float(liste_distances[i])   #  Conversion de la distance si elle est
de type int.
        point_precedent = (liste_x[i], liste_y[i])  #  Charge le point précédent.
        point_suivant = t(ma_norme, point_precedent)  #  Récupère le point suivant calc
ulé par t.
        liste_noms.append(mon_nom)  #  Création du nom du point suivant.
        liste_x.append(point_suivant[0])  #  Calcul de son abscisse.
        liste_y.append(point_suivant[1])  #  Calcul de son ordonnée.
    liste_noms[-1] ='A'  #  Le dernier point est A (Arrivée).


    # Création de la figure fig et de l'objet ax
    fig = plt.figure(figsize=(12, 12))  #  Création d'un objet de la classe Figure en p
```

```
récisant sa taille.
    ax = plt.axes()  #  Création d'un objet de la classe Axes

    # Affichage des points représentatnt les stations.
    ax.plot(liste_x, liste_y) # Affichage des segments
    ax.set_ylim(600, -50)  #  pour que l'axe des y soit dirigé à l'envers.


    # Affichage des points représentant les stations
    ax.scatter(liste_x, liste_y, s = 30, c = 'blue')


    # Affichage des noms des points
    for i, txt in enumerate(liste_noms):
        ax.annotate(txt, (liste_x[i]+5, liste_y[i]-5))  #  i est l'indice et txt est l'
élément


    #  Calcul des coordonnées des milieux pour positionner les distances.
    liste_x_milieux = []
    liste_y_milieux = []
    for i in range(len(liste_distances)):
        x, y = (liste_x[i+1] + liste_x[i])/2, (liste_y[i+1] + liste_y[i])/2
        x = round(x)  # Arrondit à l'entier
        y = round(y)  # Arrondit à l'entier
        liste_x_milieux.append(x)
        liste_y_milieux.append(y)


    # Ajout des étiquettes des distances au niveau des milieux des segments
    for i, txt in enumerate(liste_distances):  #  i est l'indice et txt est l'élément.
        ax.annotate(txt, (liste_x_milieux[i]-20, liste_y_milieux[i]+20))  #  Positionne
les distances un
                                                                          #  peu à côté
des milieux.

    # Taille du graphe
    xmin = min(liste_x)
    ymin = min(liste_y)
    xmax = max(liste_x)
    ymax = max(liste_y)
    ax.set(xlim=(xmin - 50, xmax + 50), ylim=(ymax + 50, ymin - 50))  #  Pour que A et
D ne soient sur les coins.
    ax.set_aspect('equal')  #  pour que l'échelle soit la même sur les deux axes

    plt.show()

ma_liste = [137, 96, 105, 128, 132, 88, 76, 51]
affiche_figure(ma_liste)
```

48) Complétez ci-dessous la fonction glouton. On remarque que cet algorithme glouton prend, comme le précédent, deux paramètres.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectation
de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:

```
def glouton(distances, dmax):
    """
    Calcule la liste des stations où l'automobiliste doit faire le plein.

    Paramètres nommés
    -----------------
        distances : de type list
                    La liste des distances entre les stations.
        dmax : de type float
               L'autonomie maximale en distance de la voiture.

     Retourne
     --------
         stations : de type list
                    La liste des stations où le plein doit être fait.

    """


    #  n est le nombre de distances entre les stations. C'est aussi le nombre de statio
ns en comptant l'arrivée.
    ## n =
    #  d est l'autonomie qu'il reste à la voiture. Au départ l'autonomie égale l'autono
mie max (réservoir plein).
    ## d =
    # Initialisation de la liste stations et de l'index qui sert à parcourir la liste.
    ## stations =
    ## i =
    while i != n:   #  Le dernier indice dans la liste des distances est n - 1.
        while i < n and distances[i] <= d:   # On reste dans la boucle si la prochaine
distance est
                                             # inférieure à l'autonomie restante d.
            #  On actualise l'autonomie restante en enlevant les kms de l'étape.
            ## d =
            #  On prépare l'indice pour le tour de boucle while suivant.
            ## i =
        # Lorsqu'on est sorti de la boucle while interne, on s'est arrêté faire le plei
n. On ajoute la station.
        stations.append(i - 1)
        # L'autonomie égale à nouveau l'autonomie max (réservoir plein).
        ## d =
    return stations
```

- Un premier test :

In [ ]:
```
distances_interstations = [137, 96, 105, 128, 132, 88, 76, 51]

mes_arrets = glouton(distances_interstations, 400)
print(mes_arrets)
```

- Vous avez trouvé que l'automobiliste doit s'arrêter dans les stations 2 et 5 (et bien entendu à la 7 qui est l'arrivée) ? Parfait ! Vous pouvez continuer...

- Exécutez le codes suivant pour visualiser les stations (en rouge) où doit s'arrêter l'automobiliste :

In [ ]:
```
#  Importation des bibliotheques permettant les graphiques
%matplotlib inline
import matplotlib.pyplot as plt   # Importation pour dessiner.
import math  #  Importation pour avoir les fonctions cos et sin.
import random  #  Importation pour avoir un angle au hasard

#  Definition de la liste des distances


# Création de la fonction translation du point précédente au suivant
def t(r, p):
    """
    Translate le point p d'un vecteur de norme r et de direction aléatoire

    Paramètres nommés
    -----------------
        r : de type float
            La longueur du vecteur de la translation.
        p : de type tuple de nombres de type float
            Les coordonnées (x, y) du point.

     Retourne
     --------
         x, y : de type tuple de nombres de type float
                Le coordonnées du point image par la translation.

    """
    theta = float(random.randint(0, 90))
    theta = theta * math.pi/180  # Conversion de l'angle aléatoire en radians.
    x = p[0] + r * math.cos(theta)
    y = p[1] + r * math.sin(theta)
    return x, y



def affiche_figure(liste_distances, arrets):
    """
    Affiche le trajet et les stations en respectant la liste de distances
    Les stations où l'automobiliste doit s'arrêter sont en rouge.

    Paramètres nommés
    -----------------
        liste_distances : de type list de nombres de type float
            La liste des distances entra stations.
        arrets : de type int
            Les numéros de stations fournies par l'algorithme glouton.

     Retourne
     --------
         Aucun
                Un graphique est tracé.

    """



    #  Construction de la liste des noms, des abscisses, des ordonnées.
    liste_noms = ['D']  #  Le premier point est D (Départ).
    liste_x = [0]  #  Initialisation avec l'abscisse de D.
```

```
    liste_y = [0]  #  Initialisation avec l'ordonnée de D.

    #  On construit le point suivant en fonction du précédent, en tenant compte de leur
distance.
    for i in range(len(liste_distances)):
        mon_nom = 'S' + str(i)  #  Création du nom du nouveau point.

        ma_norme = float(liste_distances[i])   #  Conversion de la distance si elle est
de type int.
        point_precedent = (liste_x[i], liste_y[i])  #  Charge le point précédent.
        point_suivant = t(ma_norme, point_precedent)  #  Récupère le point suivant calc
ulé par t.
        liste_noms.append(mon_nom)  #  Création du nom du point suivant.
        liste_x.append(point_suivant[0])  #  Calcul de son abscisse.
        liste_y.append(point_suivant[1])  #  Calcul de son ordonnée.
    liste_noms[-1] ='A'  #  Le dernier point est A (Arrivée).


    # Création de la figure fig et de l'objet ax
    fig = plt.figure(figsize=(12, 12))  #  Création d'un objet de la classe Figure en p
récisant sa taille.
    ax = plt.axes()  #  Création d'un objet de la classe Axes

    # Affichage des points représentatnt les stations.
    ax.plot(liste_x, liste_y) # Affichage des segments
    ax.set_ylim(600, -50)  #  pour que l'axe des y soit dirigé à l'envers.


    # Affichage des points représentatnt les stations. Les stations où il faut s'arrête
r sont en rouge.
    colors = ['blue']  #  Cette liste contient des éléments 'red' si l'indice i est dan
s la liste des arrêts.
                       #  Elle contient des éléments 'blue' sinon. Le point de départ D
est toujours bleu.
    for i in range(len(liste_distances)):
        if i in arrets:
            colors.append('red')
        else:
            colors.append('blue')

    ax.scatter(liste_x, liste_y, s = 30, c = colors, picker = "True")  #  La fonction d
essin de points
                                                                       # prend les coule
urs dans colors

    # Affichage des noms des points
    for i, txt in enumerate(liste_noms):
        ax.annotate(txt, (liste_x[i]+5, liste_y[i]-5))  #  i est l'indice et txt est l'
élément


    #  Calcul des ccordonnées des milieux pour positionner les distances.
    liste_x_milieux = []
    liste_y_milieux = []
    for i in range(len(liste_distances)):
        x, y = (liste_x[i+1] + liste_x[i])/2, (liste_y[i+1] + liste_y[i])/2
        x = round(x)  # Arrondit à l'entier
        y = round(y)  # Arrondit à l'entier
        liste_x_milieux.append(x)
        liste_y_milieux.append(y)
```

```
    # Ajout des étiquettes des distances au milieux des segments
    for i, txt in enumerate(liste_distances):  #  i est l'indice et txt est l'élément.
        ax.annotate(txt, (liste_x_milieux[i]-20, liste_y_milieux[i]+20))  #  Positionne
les distances un
                                                                          #  peu à côté
des milieux.

    # Taille du graphe
    xmin = min(liste_x)
    ymin = min(liste_y)
    xmax = max(liste_x)
    ymax = max(liste_y)
    ax.set(xlim=(xmin - 50, xmax + 50), ylim=(ymax + 50, ymin - 50))  #  Pour que A et
D ne soient sur les coins.
    ax.set_aspect('equal')  #  pour que l'échelle soit la même sur les deux axes

    plt.show()

ma_liste = [137, 96, 105, 128, 132, 88, 76, 51]

mes_arrets = glouton(ma_liste, 400)
affiche_figure(ma_liste, mes_arrets)
```

49) Dans la cellule ci-dessous, complétez le programme. Ce programme va fournir une liste aléatoire de distances séparant deux stations le long d'un trajet.

- L'utilisateur fournit la distance totale du trajet et l'autonomie de la voiture.
- Le programme affiche les stations où l'automobiliste devra s'arrêter faire le plein.

```
# indique un commentaire comme d'habitude.

## indique qu'il y a quelque chose à écrire ou à compléter (une fonction, une affectat
ion de variable etc). Vous supprimez ensuite les deux ##.
```

In [ ]:
```
#  Importation de la fonction qui fournit un entier aléatoire entre deux bornes.
from random import randint
trajet = 2300 # km
# Initialisation du tableau des distances inter stations avec une première distance alé
atoire.
## tab =
while sum(tab) < trajet:
    #  Remplissage du tableau des distances inter stations jusqu'à la limite du trajet
total.
    tab.append(randint(25, 100))
#  On écrit la dernière distance de façon à avoir un trajet total d'exactement 2300 km.
## tab[-1] =
res = 600  #  600 kms avec le plein du réservoir

# Affiche les distances inter stations
print("Les distances inter stations sont:")
print(tab)

# Affiche la liste des stations où il faut faire le plein, indiquées par la fonction gl
outon.
print("Les stations où l'automobiliste doit s'arrêter sont les numéros :")
print(glouton(tab, res))
```

- Le programme fonctionne ?

Parfait ! On va pouvoir visualiser le trajet correspondant. Exécutez le code ci-dessous :

In [ ]:
```
#  Importation des bibliotheques permettant les graphiques
%matplotlib inline
import matplotlib.pyplot as plt   # Importation pour dessiner.
import math  #  Importation pour avoir les fonctions cos et sin.
import random  #  Importation pour avoir un angle au hasard

#  Definition de la liste des distances


# Création de la fonction translation du point précédente au suivant
def t(r, p):
    """
    Translate le point p d'un vecteur de norme r et de direction aléatoire

    Paramètres nommés
    -----------------
        r : de type float
            La longueur du vecteur de la translation.
        p : de type tuple de nombres de type float
            Les coordonnées (x, y) du point.

     Retourne
     --------
         x, y : de type tuple de nombres de type float
                Le coordonnées du point image par la translation.

    """
    theta = float(random.randint(0, 90))
    theta = theta * math.pi/180  # Conversion de l'angle aléatoire en radians.
    x = p[0] + r * math.cos(theta)
    y = p[1] + r * math.sin(theta)
    return x, y



def affiche_figure(liste_distances, arrets):
    """
    Affiche le trajet et les stations en respectant la liste de distances
    Les stations où l'automobiliste doit s'arrêter sont en rouge.

    Paramètres nommés
    -----------------
        liste_distances : de type list de nombres de type float
            La liste des distances entra stations.
        arrets : de type int
            Les numéros de stations fournies par l'algorithme glouton.

     Retourne
     --------
         Aucun
                Un graphique est tracé.

    """



    #  Construction de la liste des noms, des abscisses, des ordonnées.
    liste_noms = ['D']  #  Le premier point est D (Départ).
    liste_x = [0]  #  Initialisation avec l'abscisse de D.
```

```
    liste_y = [0]  #  Initialisation avec l'ordonnée de D.

    #  On construit le point suivant en fonction du précédent, en tenant compte de leur
distance.
    for i in range(len(liste_distances)):
        mon_nom = 'S' + str(i)  #  Création du nom du nouveau point.

        ma_norme = float(liste_distances[i])   #  Conversion de la distance si elle est
de type int.
        point_precedent = (liste_x[i], liste_y[i])  #  Charge le point précédent.
        point_suivant = t(ma_norme, point_precedent)  #  Récupère le point suivant calc
ulé par t.
        liste_noms.append(mon_nom)  #  Création du nom du point suivant.
        liste_x.append(point_suivant[0])  #  Calcul de son abscisse.
        liste_y.append(point_suivant[1])  #  Calcul de son ordonnée.
    liste_noms[-1] ='A'  #  Le dernier point est A (Arrivée).


    # Création de la figure fig et de l'objet ax
    fig = plt.figure(figsize=(16, 16))  #  Création d'un objet de la classe Figure en p
récisant sa taille.
    ax = plt.axes()  #  Création d'un objet de la classe Axes

    # Affichage des points représentatnt les stations.
    ax.plot(liste_x, liste_y) # Affichage des segments
    ax.set_ylim(600, -50)  #  pour que l'axe des y soit dirigé à l'envers.


    # Affichage des points représentatnt les stations. Les stations où il faut s'arrête
r sont en rouge.
    colors = ['blue']  #  Cette liste contient des éléments 'red' si l'indice i est dan
s la liste des arrêts.
                       #  Elle contient des éléments 'blue' sinon. Le point de départ D
est toujours bleu.
    for i in range(len(liste_distances)):
        if i in arrets:
            colors.append('red')
        else:
            colors.append('blue')

    ax.scatter(liste_x, liste_y, s = 30, c = colors, picker = "True")  #  La fonction d
essin de points
                                                                       # prend les coule
urs dans colors

    # Affichage des noms des points
    for i, txt in enumerate(liste_noms):
        ax.annotate(txt, (liste_x[i]+5, liste_y[i]-5))  #  i est l'indice et txt est l'
élément


    #  Calcul des coordonnées des milieux pour positionner les distances.
    liste_x_milieux = []
    liste_y_milieux = []
    for i in range(len(liste_distances)):
        x, y = (liste_x[i+1] + liste_x[i])/2, (liste_y[i+1] + liste_y[i])/2
        x = round(x)  # Arrondit à l'entier
        y = round(y)  # Arrondit à l'entier
        liste_x_milieux.append(x)
        liste_y_milieux.append(y)
```

```
    # Ajout des étiquettes des distances au milieux des segments
    for i, txt in enumerate(liste_distances):  #  i est l'indice et txt est l'élément.
        ax.annotate(txt, (liste_x_milieux[i]-20, liste_y_milieux[i]+20))  #  Positionne
les distances un
                                                                          #  peu à côté
des milieux.

    # Taille du graphe
    xmin = min(liste_x)
    ymin = min(liste_y)
    xmax = max(liste_x)
    ymax = max(liste_y)
    ax.set(xlim=(xmin - 50, xmax + 50), ylim=(ymax + 50, ymin - 50))  #  Pour que A et
D ne soient sur les coins.
    ax.set_aspect('equal')  #  pour que l'échelle soit la même sur les deux axes


    # Ajout de texte sur la Figure à une position arbitraire
    ax.text(xmax/2 - 100, 150, 'Distance totale du trajet : 2300 km \n\n Autonomie : 60
0 km \n\n Nombre \
    total de stations : '+str(glouton(tab,res)[-1]), fontsize = 15,
           bbox={'facecolor': 'red', 'alpha': 0.5, 'pad': 10})
    ax.text

    plt.show()



mes_arrets = glouton(tab, res)  #  La liste des arrêts
affiche_figure(tab, mes_arrets)  #  La liste des distances aléatoires calculées par le
programme de lap. 327
```

- En résumé :
  - Un algorithme glouton ne donne pas en général la solution optimale.
  - Cependant, dans certains cas comme celui des stations d'essence, on démontre qu'il donne la solution optimale.

Pour finir, visionnez la vidéo ci-dessous résume les algorithmes glouton avec un exemple déjà vu dans ce cours et un autre exemple.

(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/algorithme_glouton.mp4)