Activité 1 : structures de données.

Notion de **s**tructure de **d**onnées **a**bstraite : sda
Notion d'interface. Implémentation d'une sda : la **s**tructure de **d**onnées **c**oncrètes (sdc).

E. Bansière- P. Jonin

1.Mise en situation

Dans le transport aérien, lorsque l'aéroport est congestionné à l'arrivée des appareils, les contrôleurs aériens les placent avant l'atterrissage dans un circuit d'attente pour les faire patienter.

Ce circuit aussi appelé hippodrome, permet de préparer l'avion à l'atterrissage et de séparer les avions à l'arrivée dans les différents terminaux. On trouve donc toujours dans un aéroport un ou plusieurs circuits d'attentes où plusieurs appareils peuvent être "stockés".

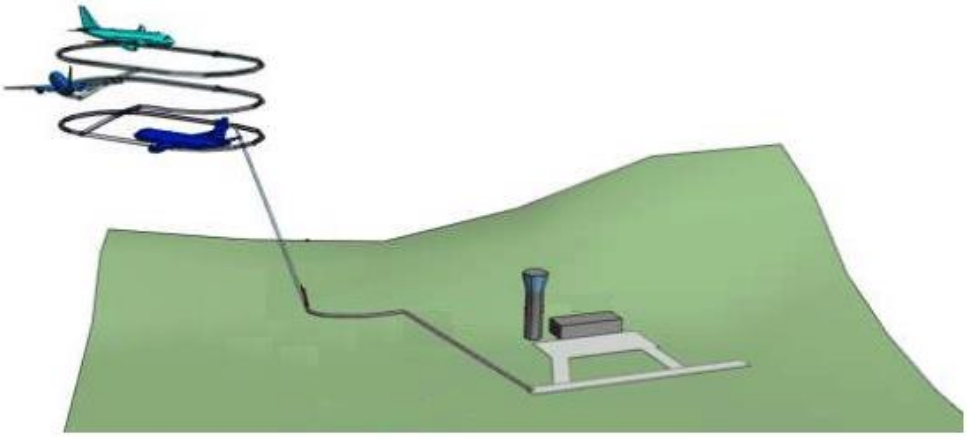
Les avions placés en attente réalisent des trajectoires en forme d'hippodromes, chaque avion avec une altitude différente. Lorsque les conditions d'atterrissage deviennent possibles (piste disponible), le contrôleur demande aux pilotes de l'avion ayant la plus faible altitude de sortir de la file pour atterrir.

Les situations peuvent aussi devenir prioritaires avec des conditions météorologiques dégradées ou des avions présentant des urgences (un avion avec un niveau bas de carburant par exemple).

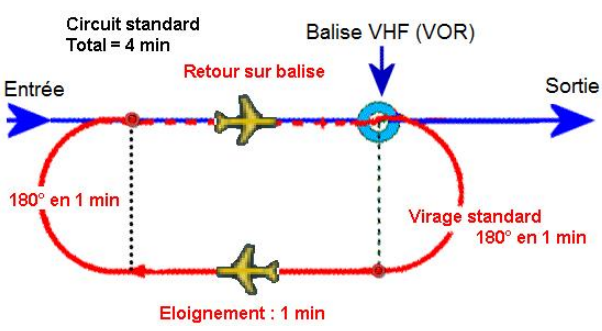
Données concernant le circuit d'attente

Temps moyen d'un tour d'attente	4 minutes
Temps moyen pour sortir de la file, atterrir et dégager la piste	10 minutes
Nombre d'avions maximums dans une file d'attente	10

Trois appareils dans le circuit d'attente avant l'atterrissage

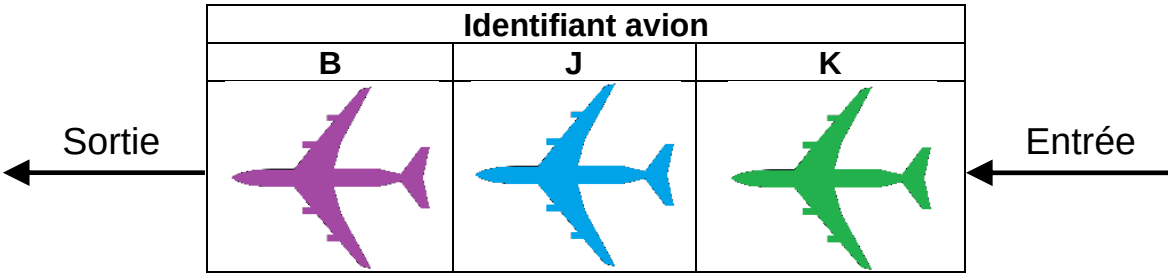


L'appareil autorisé par le contrôleur aérien sort du circuit d'attente et peut atterrir

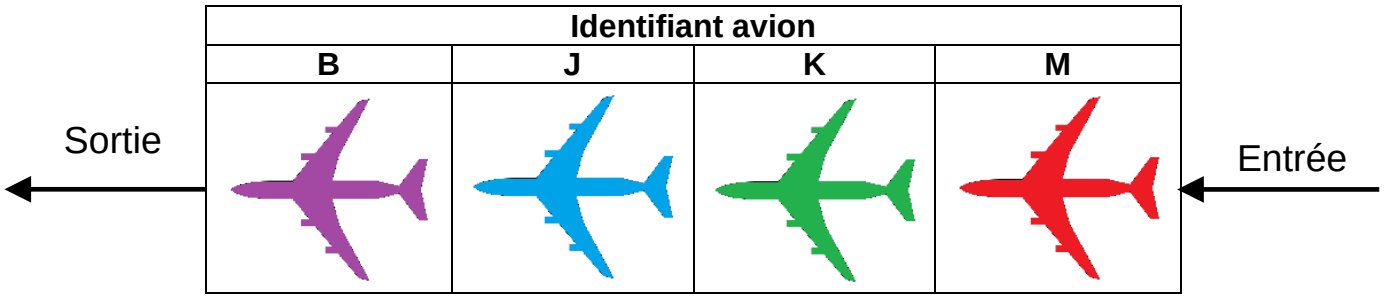


2. La notion de file d'attente

Supposons qu'il y ait 3 avions en attente d'atterrissage.

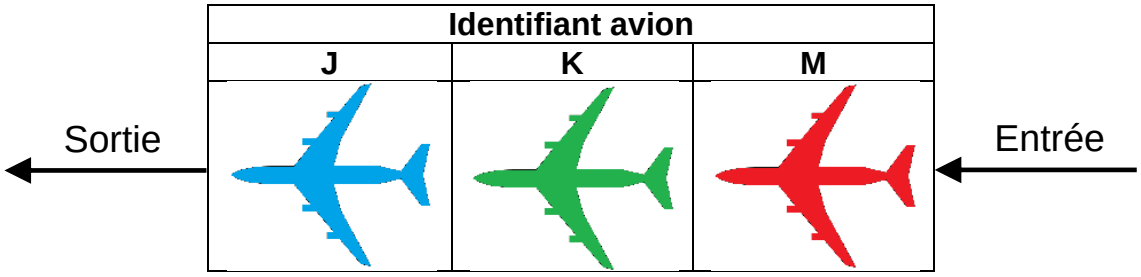


Un nouvel avion M arrive en provenance de Hong-Kong. Celui-ci entre dans la file d'attente. L'opération correspondante en algorithmique s'appelle : **enfiler(F, 'M').**



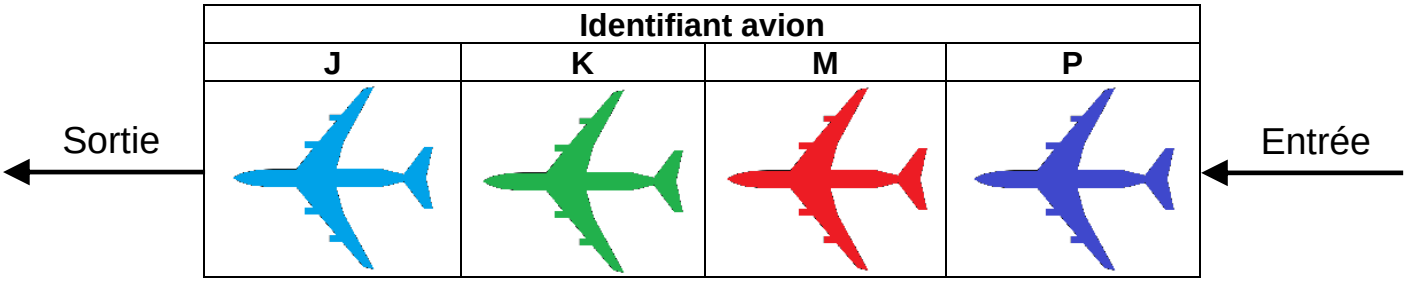
L'avion B est autorisé à atterrir. Celui-ci sort de la file.

L'opération correspondante en algorithmique s'appelle : **defiler(F).**



Un nouvel avion P arrive en provenance de Hambourg. Celui-ci entre dans la file d'attente.

L'opération correspondante est donc cette fois : **enfiler(F, 'P').**



Le contrôleur de vol doit connaitre le nombre d'avions dans la file d'attente. L'opération effectuée en algorithmique est **longueur(F)** qui retourne le nombre d'avions dans la file.

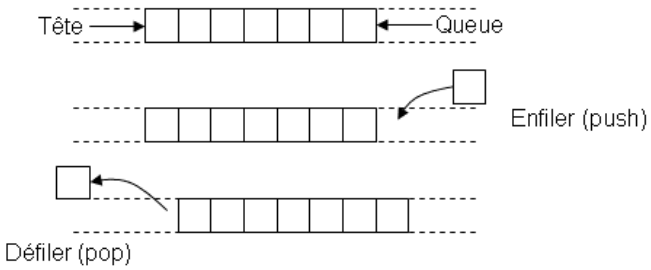
Le contrôleur de vol peut également savoir s'il n'y a plus d'avions dans la file d'attente. L'opération effectuée en algorithmique est estVide(F) qui retourne Vrai si la file est vide.

3. A la main …

On part d'une file F vide …

a) Que contiendra cette file après les opérations suivantes ? :

enfiler(F, 'L') …
enfiler(F, 'B') …
enfiler(F, 'A') …
defiler(F) …
enfiler(F,'P') …
defiler(F) …
defiler(F) …
enfiler(F,'K') …



b) Que retourne alors l'instruction : longueur(F) ?

4. L'interface de notre **s**tructure de **d**onnées **a**bstraite (sda)

- **Représentation : <'a', 'b', 'c' <**

L'élément 'a' est le premier de la file, l'élément 'c' est le dernier. Ainsi, les premiers éléments ajoutés à la file seront les premiers à en être retirés.

- **Opérations :**

-
-
-
-
-

Cette structure est du type FIFO (First In, First Out : premier entré, premier sorti)

5. Une première implémentation en python en utilisant une chaine de caractères …

Nous allons nous limiter ici à une file comportant 5 éléments au maximum : le programme devra retourner un message indiquant que la file est pleine au-delà de cette limite.

Nous devons créer pour chacune des opérations de la sda une fonction. Nous obtiendrons alors une structure de données **c**oncrète (une sd**c**).

Nous allons pour cela utiliser une chaine de caractères.

Nous devons donc construire 5 fonctions :

```
def creer():
    return("")
def longueur(file):
def enfiler (file, element):
def defiler (file):
def estVide(file):
#Programme principal
attente=creer()
attente=enfiler(attente,'L')
print(attente)
print(longueur(attente))
```

Eléments pour le codage : à tester en mode console :

```
chaine='ABCD'
len(chaine)
chaine[1:]
chaine[:-1]
chaine=chaine+'J'
```

Remarques :

Lorsqu'on appelle nos fonctions il faut toujours utiliser une affectation du type : attente=enfiler(attente,'L')

En effet il faut se souvenir que le type String en Python n'est pas mutable : lorsqu'on définit une chaine, on ne peut plus la modifier ! La seule façon de le faire est de la réaffecter, c'est-à-dire la relier en mémoire à un nouvel objet…

6. Une seconde implémentation : en utilisant un tableau

Un gros défaut de l'implémentation précédente est le format de l'identification de l'avion restreint à une unique lettre.

En pratique, un identifiant d'avion est plutôt du type : LH713, HP856

(https://fr.wikipedia.org/wiki/Liste_des_pr%C3%A9fixes_OACI_d%27immatriculation_des_a%C3%A9ronefs)

Nous allons donc utiliser un tableau avec les règles suivantes :

- La première case du tableau (d'indice 0) contient l'indice de la tête de la file.
- La seconde case (d'indice 1) contient l'indice de la queue de la file : c'est-à-dire l'indice de la prochaine case disponible de la file.
- La troisième case du tableau (d'indice 2) contient le nombre d'éléments présents dans la file.
- A chaque fois que l'on enfile un élément, on incrémente la taille d'une unité ainsi que l'indice de la queue.
- A chaque fois que l'on défile un élément, on décrémente la taille d'une unité et on incrémente l'indice de la tête d'une unité.
- **Dès que les indices de tête ou de queue dépasse la longueur maximale du tableau, ils repartent au début du tableau (indice 3) ; on appelle cela un tableau avec une gestion circulaire (nous allons comprendre ceci à la page suivante !)**
- Lorsqu' un élément est défilé il reste dans le tableau mais il n'est plus dans la file ! (car l'indice de tête est décalé à droite !). Il sera éventuellement écrasé par un retour circulaire !
- Il est tout à fait possible que l'indice de queue soit plus petit que l'indice de tête !

Observons le contenu de la file et notre tableau qui l'implémente sur l'exemple suivant (on travaille toujours avec une file de taille maximale 5)

Opération	Contenu file	Implémentation dans le tableau
Créer(F)		
enfiler(F, 'LH713')		
enfiler(F, 'HP856')		
enfiler(F, 'FR174')		
defiler(F)		
enfiler(F,'AF644')		
defiler(F)		
defiler(F)		
enfiler(F,'KL201')		
enfiler(F,'FR122')		
enfiler(F,'FR999')		
Defiler(F)		
Defiler(F)		
Defiler(F)		
Defiler(F)		

Codage Python correspondant : 4 fonctions à compléter !

```
def creer():
   tab=[0]*8
   tab[0]= …
   tab[1]= …
   return(tab)

def longueur(file):
   return

def enfiler (file, element): # Prévoir le cas file pleine et du retour
                                circulaire …


def defiler (file): # Prévoir file vide et retour circulaire ...
```

Pour tester votre code :

```
attente=creer()
print(attente)
enfiler(attente,'LH713')
print(attente)
enfiler(attente,'HP856')
print(attente)
enfiler(attente,'FR174')
print(attente)
defiler(attente)
print(attente)
…
```

Remarques importantes :

1) **On observe cette fois dans les appels qu'il n'y a pas d'affectation lorsqu'on utilise enfiler et defiler. Les fonctions agissent ici comme des procédures et modifient le paramètre envoyé ! (D'ailleurs nous n'avons pas mis de return …)**

C'est dangereux et cela doit être bien analysé : le type tableau en Python (des listes en fait …) est mutable !
Lorsqu'on passe en paramètre un tableau (variable globale 'attente'), la variable locale qui le récupère dans la fonction ('file') pointe en réalité sur le même espace mémoire que la variable 'attente'. Tout action sur 'file' (possible car mutable) va modifier aussi 'attente' !
On pourrait améliorer ceci en créant dans la fonction une copie réelle de la variable tableau et retourner cette copie modifiée… C'est un peu lourd mais ce serait plus propre !

2) **Le type list en Python n'est pas vraiment un tableau au sens strict car il est dynamique. De plus il existe des méthodes toutes faites permettant de mimer une file …**

7. Complément pour coder en Python.

On peut en fait coder très rapidement une file en Python, en utilisant deux méthodes sur les listes :

- **pop ()**
- **append()**

1) Tester en mode console :

a) Enfiler …

```
>>> attente=[7,11,4,75,124]
>>> attente.append(666)
>>> print(attente)
```

b) Défiler …

```
>>> attente=[7,11,4,75,124]
>>> attente.pop(0)
>>> print(attente)
```

2) Proposer alors un codage utilisant ces deux méthodes

Conclusion :

Nous avons codé la sda file de quatre façons :

- Avec une chaine de caractères (un string) ! mais les avions avaient un identifiant sous la forme d'une unique lettre.
- Avec un tableau tournant de taille 8 (version procédurale + version fonctionnelle …)
- Avec un tableau et deux méthodes spécialisées : append() et pop()

Bien s'assurer que vous disposez de toutes ces versions dans un dossier dédié …

8. Prolongement : une piste pour le grand oral ?

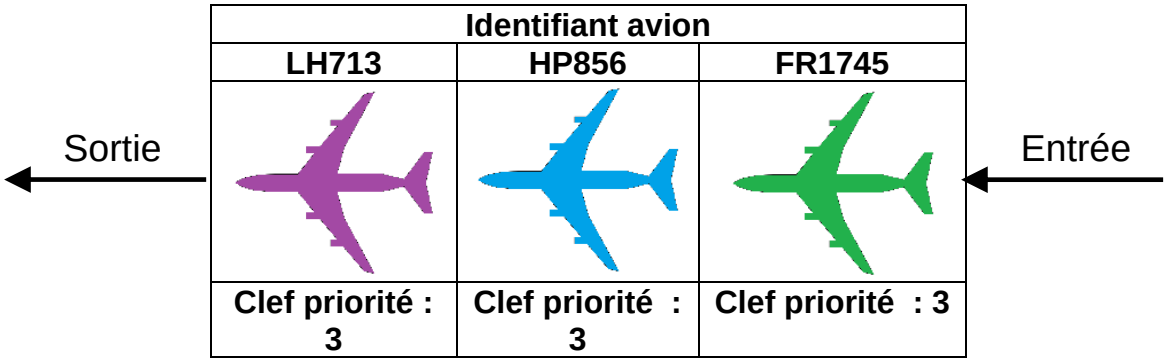
Dans le cas d'une file d'attente avec priorité, les avions ont maintenant un ordre de priorité. En effet, il se peut qu'un avion ait peu de carburant restant par exemple et doive atterrir avant les autres.

Le niveau de priorité est défini par une clef où la priorité la plus importante correspond à la valeur la plus petite.

Les nouvelles contraintes de la file sont donc les suivantes :

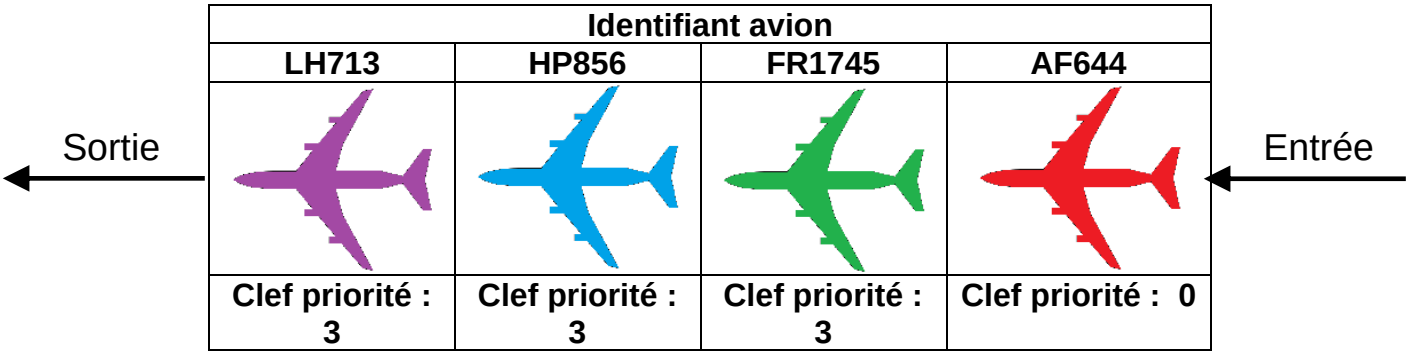
	Clef de priorité	**Information**
Tout va bien dans l'avion, il reste 1H de carburant	3	Avion non prioritaire
Il reste 30 minutes de carburant	2	
Il reste 15 minutes de carburant	1	
Panne sèche, l'avion plane	0	Avion prioritaire devant tous les autres

Supposons qu'il y ait 3 avions en attente d'atterrissage de même priorité dans la file d'attente.



Le 1er avion autorisé à atterrir dans ce cas sera le numéro LH713. En effet, c'est le premier de la file FIFO (premier entré, premier sorti).

Si un nouvel avion AF644 arrive en provenance de Hong-Kong avec une priorité maximale, il entre dans la file d'attente avec une clef de priorité 0. L'opération est enfiler(0,'AF644').



Dans ce cas, le 1er avion autorisé à atterrir sera maintenant le numéro AF644 en raison de sa clef de priorité 0.

Analyser en quoi ces nouvelles conditions modifient l'interface de la sda et l'implémenter en Python !