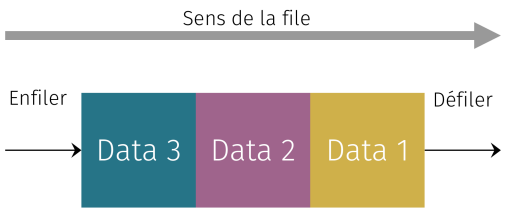
Chapitre 1

Files

1 Principe



On part d'une file vide, on enfile les éléments un par un.

On ne peut que retirer le premier élément enfilé : « Premier entré, premier sorti », *First in first out* (FIFO).

Applications

- File d'impression de documents.
- File d'évènements dans `pygame` ou en général en programmation *évènementielle*.
- Algorithmes de gestion des stocks.
- Parcours de graphes et d'arbres.
- D'autres applications sont données en exercice.

1.1 Interface de la structure de données file

Elle est très simple !

Interface de la pile

- *file_vide()* créée une file vide ;
- *enfiler(pile,valeur)* enfile la valeur dans la file ;

- *defiler(pile)* renvoie le prochain élément dans la file et l'enlève ;
- *est_vide()* indique si la file est vide on non ;

1.2 Implémentations de l'interface

- Avec une simple liste python.
- Avec une liste encapsulée dans un objet.
- Avec 2 piles !

2 Exercices

Exercice 1 : Implémentation objet

Écrire une classe `Queue` qui implémente la structure de file vue en cours. Sauvegarder dans un module `queue.py`. Celui-ci servira pour les exercices suivants.

Vous pouvez nommer les méthodes `enqueue`, `dequeue` et `is_empty`.

Exercice 2 : 2 piles pour une file

Écrire une classe `QueueWith2Stacks` qui implémente la structure de file avec 2 piles, comme sur le schéma du cours.

Exercice 3 : Maximum d'une file

Écrire une fonction `max_queue` qui

- en entrée prend une file composée d'`int` ;
- renvoie le maximum de cette file. Attention la file doit être remise dans l'état initial.

Exercice 4 : Simulation de passage à la caisse

Une grande surface possède n caisses. On choisit une unité de temps arbitraire (le tour) et on décide que lorsqu'un client passe à la caisse, cela prend un temps aléatoire compris entre 1 et n unités de temps.
À chaque unité de temps, un client arrive aux caisses.

On commence la simulation avec n clients qui arrivent.
On aimerait simuler le temps d'attente aux caisses et évaluer le temps d'attente moyen par client.

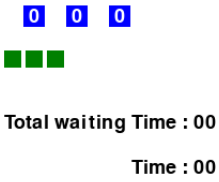
Avec une seule file

On décide qu'une seule file existe : les clients attendent dans la file et dès qu'une caisse se libère, le premier client dans la file est reçu.

Simulation

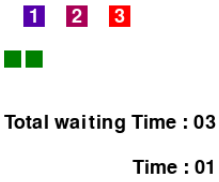
Voici un début de simulation avec n=3 caisses

Initialisation



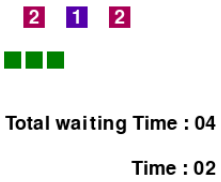
L'heure est `00`.
Puisqu'il y a 3 caisses, on place 3 clients dans la file (les carrés verts).
Pour l'instant personne n'a attendu donc le total des temps d'attente est `00`.
Les trois caisses sont libres : elles sont bleues, avec un temps d'attente de zéro.

Première itération



L'horloge a fait un tour.
Les 3 clients sont passés en caisse : les deux dernières prendront 3 tours pour traiter les achats de son client, la première 2 tours.
Un nouveau client se présente et attend.

Deuxième itération



L'horloge a fait un tour.
Un nouveau client arrive dans la file.
Il est obligé d'attendre lui aussi. Pour l'instant le temps d'attente total n'est pas actualisé : on attend qu'un client soit servi avant de comptabiliser son temps d'attente.

Troisième itération

L'horloge a fait un tour.
Le client qui attendait depuis 2 tours est reçu, on actualise le temps d'attente total. Il passe dans la première caisse. Au total, 4 clients ont été reçus, avec un temps d'attente total de 2 tours, donc un temps d'attente moyen de 0.5 tour par client.
Un nouveau client se présente dans la file.

Quatrième itération

L'horloge a fait un tour.
Les 2 clients précédents sont servis, on ajoute leurs temps d'attente, un nouveau client arrive, *et cætera*.

Conseils pour démarrer

File

Pour commencer on peut créer une file `file_attente` et une variable `tour` valant `0`.
Ensuite, pour se rappeler de l'heure d'arrivée d'un client il suffit d'enfiler son heure d'arrivée, c'est-à-dire la valeur de la variable `tour`.
On peut définir une constante `NB_CAISSES` et mettre en file `NB_CAISSE` clients.

Caisses

On peut créer une classe `Caisse` qui va fonctionner un peu comme la classe `Ball` déjà rencontrée, avec

- Une variable **de classe** `nb_caisses` valant 0 au départ ;
- Une variable **de classe** `caisses` de type `list`, valant `[]` au départ, pour stocker les différentes caisses ;
- Une variable **de classe** `nb_clients_servis` valant 0 ;
- Une méthode `__init__` qui crée des instances de classes avec (au minimum) les attributs suivants :
  - `self.file`, la file d'attente, passée en paramètre dans les constructeur (ainsi dans les prochaines partie, chaque caisse pourra avoir sa propre file) ;
  - `self.temps_attente`, un `int` mesurant le nombre de tours avant que la caisse soit libre ;

  Quand le constructeur est appelé, `Caisse.nb_caisses` augmente de 1 et l'instance (`self`) est ajoutée à la liste `Caisse.caisses`.
- Une méthode `sert_client` qui
  - commence par enlever un client de sa file : alors on récupère son heure d'arrivée (notons la `heure`) dans la file et `tour-heure` nous donne son temps d'attente, qu'on peut ajouter à la variable `Caisse.temps_attente_total` ;
  - comme on sert un nouveau client, on peut incrémenter `Caisse.nb_clients_servis`
  - le temps passé à s'occuper du client est `randint(1,NB_CAISSES)` et devient la nouvelle valeur de l'attribut `temps_attente` de la caisse.
- Une méthode `actualise` qui sera plus tard appelée une fois par tour et :
  - enlèvera 1 au temps d'attente de la caisse ;
  - si elle est libre, servira un client (s'il y en a dans la file) ;

Boucle principale

On pourra créer `NB_CAISSES` caisses, créer une constante `NB_TOURS` et boucler sur la variable `tour` : tant qu'on a pas atteint `NB_TOURS`, on

- fait arriver un client dans la file ;
- parcourt la liste `Caisse.caisses` ;
- actualise chaque caisse ;
- termine en ajoutant 1 à `tour`

Fin du programme

C'est à vous de jouer, vous avez tout pour calculer le temps moyen d'attente par client servi.

Plusieurs files, au hasard

Adapter le programme précédent avec une file par caisse, avec 1 client par file au départ, et les suivants arrivent en choisissant une file au hasard sans en changer.

Plusieurs files, au hasard

Adapter le programme précédent avec une file par caisse, avec 1 client par file au départ, et les suivants arrivent en choisissant une caisse libre ou une avec le moins de monde.

Bilan

Dresser le bilan des 3 méthodes.