TP08 Rendu de monnaie

Pensez à écrire votre code *en premier lieu* sur feuille pour vous entraîner aux futures épreuves écrites d'informatique.

Quoi de plus naturel quand vous allez à la boulangerie que de récupérer votre monnaie avec un minimum de pièces. Pourtant le processus permettant d'obtenir la décomposition de la somme à rendre en pièces sonnantes et trébuchantes [1] n'est pas forcément chose aisée. Nous allons essayer d'en étudier quelques rouages dans ce TP.

Partie I

Mise en jambe

1. Écrire une fonction `deux_maxima(L)` qui renvoie les deux valeurs maximales de la liste `L` de nombres (d'au moins deux éléments) donnée en entrée. Ces deux valeurs peuvent être identiques. En particulier, le comportement suivant devra être vérifié :

```
>>> deux_maxima([1,4,2,58])
(58,4)
>>> deux_maxima([42,1,42,0])
(42,42)
```

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie II

Quelques fonctions de conversions

Nous allons à présent nous intéresser à la manière dont un automate rend la monnaie (par exemple après avoir acheté votre café [2] au distributeur). Pour ce faire, il doit pouvoir communiquer avec l'utilisateur qui pense en euros alors qu'en interne, l'automate ne manipule que des centimes d'euro qui ont le bon goût d'être en nombre entier.

2. Écrire [3] deux fonctions `euros2cents(valeur_en_euros)` et `cents2euros(valeur_en_cents)` qui s'occupent des conversions entre les deux systèmes de mesure. En particulier, on prendra garde au fait que
   - `euros2cents(valeur_en_euros)` prend en entrée un nombre de type « flottant » et renvoie une valeur **entière** correspondant à la valeur en cents de l'entrée exprimée en euros (on supposera que l'entrée ne peut pas avoir plus de deux chiffres après la virgule).
   - `cents2euros(valeur_en_cents)` prend en entrée un entier (la valeur en centimes d'euros) et renvoie un flottant correspondant à la conversion en euros.

1. Le saviez-vous ? Le mot « sonnantes » se reportait au bruit que les pièces faisaient quand elles se cognaient entre elles. Et le mot « trébuchantes » a pour origine le trébuchet, qui remonte au XIV^{e} siècle. Il était utilisé pour peser des choses de faible poids comme l'or. L'expression « des pièces sonnantes et trébuchantes » signifiait donc de vraies pièces à la fois faites dans un métal pur (« sonnantes ») et non rognées sur les bords (« trébuchantes »).

2. Vous feriez mieux de prendre un chocolat chaud, c'est moins addictif...

3. La notation `euros2cents` provient du monde Unix où il existe des tas de petits utilitaires pouvant convertir un fichier d'un format en un autre. Par exemple, si l'on veut passer d'un fichier PDF à un fichier de type texte, on utilise la commande `pdf2txt` où le « 2 » se lit à l'anglaise (two) et joue sur la proximité de prononciation avec la conjonction « to » anglaise qui signifie « vers ». Sur l'exemple, on fait donc la conversion « pdf vers txt ».

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

L'automate a accès aux pièces dont les valeurs en cents sont stockées dans la liste `VALEURS` suivante, triée en ordre décroissant.

```
1 VALEURS = [200, 100, 50, 20, 10, 5, 2, 1]
```

Cette liste vous sera accessible dans vos fonctions en tant que variable globale sans qu'il soit nécessaire [4] de la déclarer comme telle ni de la redéfinir à chaque fois.

On va donc représenter une décomposition en tas de pièces par une liste d'entiers de `len(VALEURS)` éléments qui stocke à chaque indice `i` le nombre de pièces de valeurs faciales valant `VALEURS[i]`. Par exemple, la liste `pieces = [0, 0, 5, 0, 2, 1, 0, 2]` représente la décomposition en cinq pièces de 50 cents, deux de 10 cents, une de 5 cents et deux de 1 cent pour une valeur totale de 277 cents ou encore 2.77 euros. Le but global de ce TP est d'obtenir une décomposition « optimale ». Avant cela, cherchons à faire le chemin inverse, ce qui est beaucoup plus facile et permettra de faire des tests de vérification en fin de course.

3. Écrire la fonction `pieces2cents(pieces)` qui convertit une liste de nombres de pièces (de même taille que la liste `VALEURS`) en sa valeur totale en cents. Elle doit renvoyer cette dernière valeur (qui devrait être entière) et être de complexité linéaire en temps.
4. En combinant la fonction précédente et l'une des fonctions précédemment écrites, proposer une version simple (2 lignes maximum sans compter les commentaires et docstrings) de la fonction `pieces2euros(pieces)` qui fonctionne comme la précédente mais renvoie la valeur en euros.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie III

Rendu de monnaie : algorithme glouton

Il n'existe qu'une seule manière de rendre 2.77 euros avec uniquement des pièces de 1 cent. En revanche, combien en existe-t-il en n'utilisant que des pièces de 1 et 2 cents ? Quelle est la manière « optimale » de rendre ces pièces de 1 et 2 cents, c'est-à-dire celle qui minimise le nombre total de pièces à rendre ?

Le problème qui se pose est donc de minimiser le nombre de pièces rendues pour un montant fixé. Pour ce faire, on va utiliser un algorithme de type « glouton ». La méthode gloutonne vise à optimiser la résolution d'un problème en partant du principe suivant : des choix locaux optimaux, étape après étape, devraient avoir de bonnes chances de produire un résultat global optimal.

À noter qu'il n'y a pas de certitude dans le cas général et on verra plus tard que le système de pièce anglais par exemple met en défaut l'algorithme glouton, mais voyons déjà comment il fonctionne :

— on commence par créer un réceptables pour les pièces, par exemple une série de zéros de même taille que la liste `VALEURS` [5] ;

4. En fait vous *ne devez pas* utiliser autre chose que la variable `VALEURS` dans vos fonction car on peut très bien vouloir changer de système monétaire pour adopter par exemple le système anglais ou américain.
5. Rappel : utilisez la taille calculée via `len(VALEURS)` car la liste pourra être différente d'un appel à un autre.

— puis on passe en revue les pièces en commençant par la plus grosse pour voir combien on peut en rendre au maximum de chaque ;

— bien sûr, on modifie au vol la variable qui contient le nombre de cents qu'il reste encore à rendre ;

— avant de renvoyer la liste des pièces dûment complétée.

5. Écrire une fonction `cents2pieces(valeur_en_cents)` qui implémente l'algorithme précédent et renvoie la liste des pièces dont on espère qu'elle sera optimale.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Vous pouvez réfléchir pour prouver que l'algorithme fait bien ce qu'on lui demande de faire[6] à l'aide de l'invariant de boucle

$\mathscr{P}(i)$: « À l'étape i de la boucle, on a `pieces2cents(pieces`$_i$`)` + `reste`$_i$ = `valeur_en_cents` »

On prendra comme convention que i vaut -1 avant l'entrée dans la boucle.

Néanmoins, l'automate ne dispose que d'un nombre fini de pièces de chaque type pour rendre la monnaie. Le nombre de pièces restantes de chaque type est stocké dans une liste `TIROIR_CAISSE` telle que `TIROIR_CAISSE[i]` représente le nombre de pièces de valeur faciale `VALEURS[i]` restant en stock. Cette liste est aussi directement accessible et modifiable à l'intérieur de vos fonctions.

6. Adapter le code de la fonction précédente pour prendre en compte le fait que le réservoir de pièces disponibles n'est pas infini. Par exemple, s'il faut cinq pièces de 50 cents mais qu'il n'y en a que trois dans la caisse, on ne peut en rendre que trois et il faudra rajouter par exemple cinq pièces de 20 cents pour combler le manque.

   Spécifications complètes : votre fonction `cents2pieces_limitee(valeur_en_cents)` doit

   — renvoyer une liste de nombres de pièces s'il est possible de rendre la monnaie avec les stocks de pièces et mettre à jour la liste `TIROIR_CAISSE` car les pièces rendues ne sont plus disponibles lors de la prochaine utilisation de l'automate ;

   — si le rendu n'est pas possible, renvoyer `None` et laisser la liste `TIROIR_CAISSE` inchangée.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

6. C'est-à-dire obtenir une décomposition (pas forcément optimale) de la somme initiale en tas de pièces.

Partie IV

Pour aller plus loin : programmation dynamique

Un système de pièces est dit canonique si l'algorithme glouton présenté précédemment donne effectivement le résultat optimal en terme de nombre minimal de pièces à rendre. La plupart des systèmes de pièces actuels sont canoniques, ce qui justifie a posteriori la manière naturelle dont on procède pour rendre la monnaie en utilisant l'algorithme glouton, mais cela n'a pas toujours été le cas : le système anglais d'avant la réforme de 1971 ne l'était pas par exemple.

Un exemple simple. Supposons que l'on ait un système de pièces tel que `VALEURS = [4, 3, 1]` et que l'on veuille rendre 6 unités de monnaie correspondante. Montrez que l'algorithme glouton appliqué à ce système rend trois pièces alors que l'on peut facilement voir que deux seraient suffisantes (il suffit de donner les deux décompositions trouvées).

L'algorithme glouton est un algorithme de type « *top-down* », c'est-à-dire que l'on part du sommet et on soustrait au fur et à mesure pour arriver à un reste nul. La programmation dynamique sera au contraire de type « *bottom-up* », c'est-à-dire que l'on va résoudre le problème pour 1 cent, 2 cents, 3 cents, ... jusqu'au montant désiré en sauvegardant à chaque étape le résultat (liste de décomposition de pièces) dans une liste. Pour un nouveau montant N, on va utiliser les réponses précédentes en essayant de voir ce qui se passe si on ajoute une seule pièce de chaque type tout en gardant la décomposition avec le minimum de pièce au final parmi toutes celles possibles.

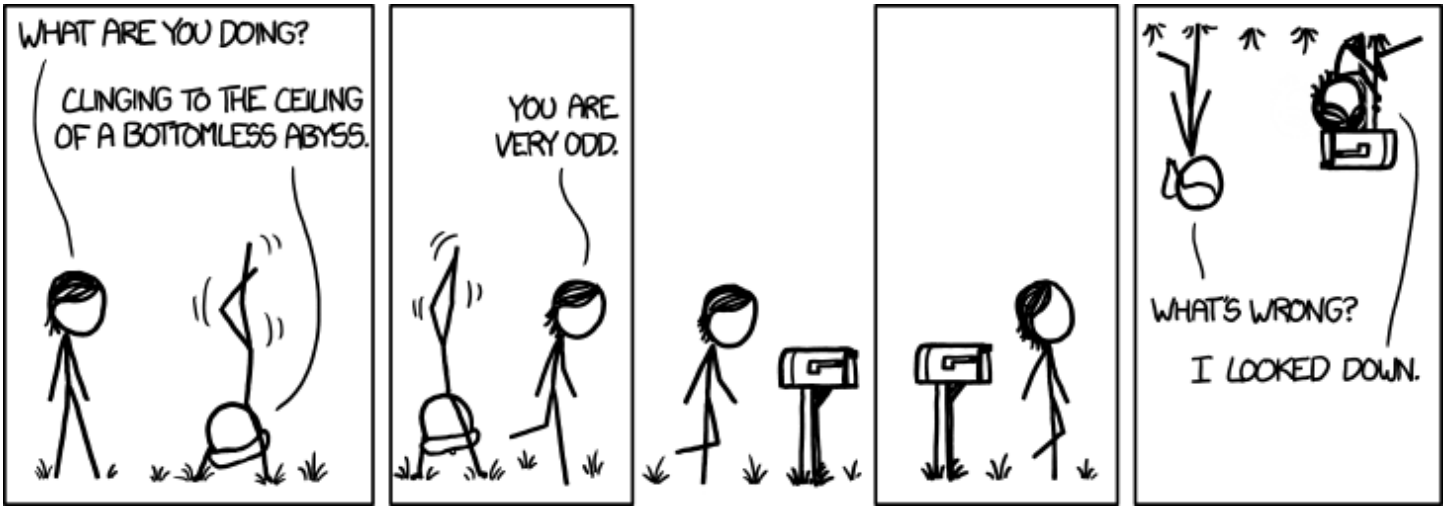
7. Écrire la fonction `bottom_up(valeur_en_cent)` qui fait un bon usage de la variable globale `VALEURS` [7] et du protocole décrit précédemment.

   NB : il n'y a plus de souci de tiroir-caisse, on a une réserve infinie de pièces.

   Hint : si vous ne trouvez pas, l'algorithme est décrit plus en détail sur la page suivante, mais laissez-vous la chance de trouver par vous-même d'abord avant d'aller voir.

Stop Gitlab

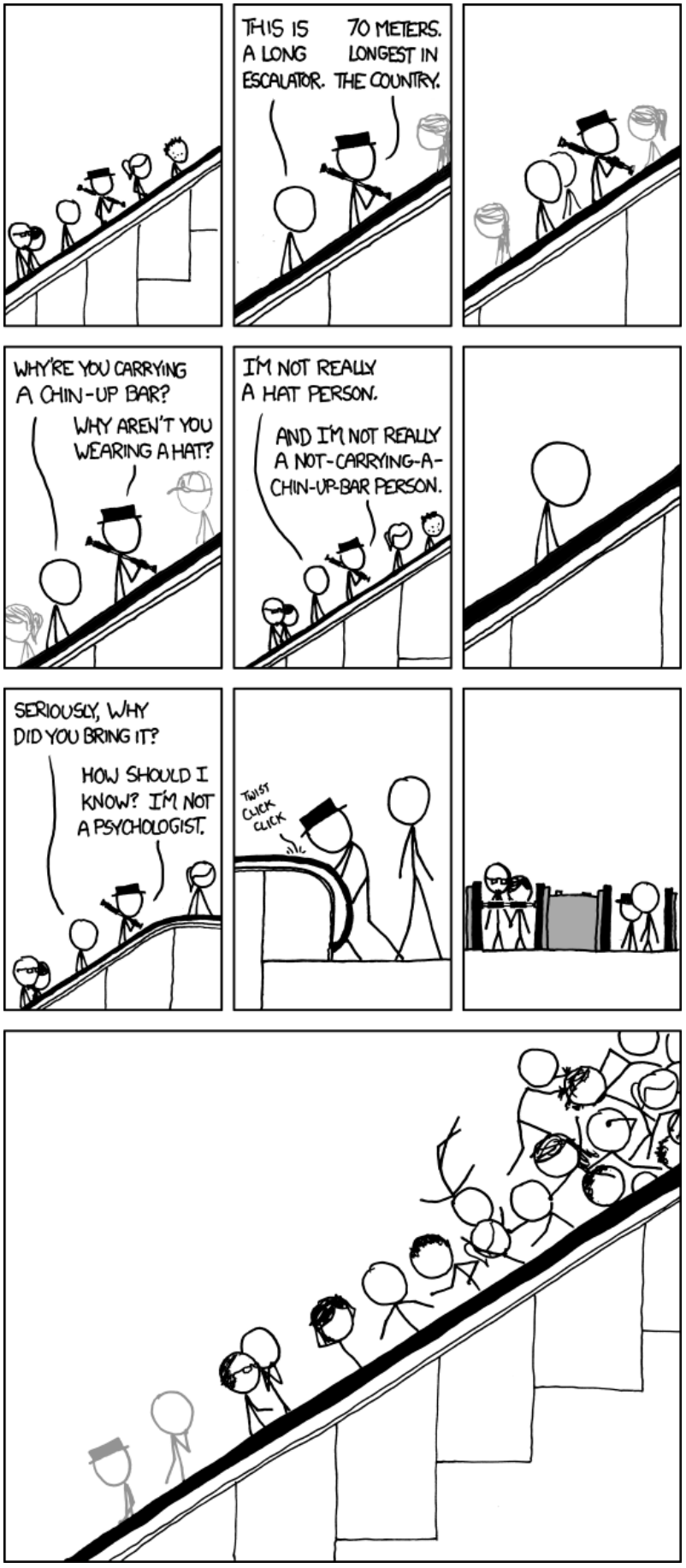
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.



.com

`I dropped a bird and I didn't hear it hit bottom.`

7. Souvenez-vous qu'on va la mettre en place pour un système non canonique, sinon ce ne serait pas drôle.



Those few who escaped found the emergency cutoff box disabled.
The stampede lasted two hours and reached the bottom three times.

Description plus explicite de l'algorithme « bottom-up » :

— Initialiser une liste « mémoire » qui contiendra les décompositions pour chaque nombre de cents. Naturellement, la position d'une décomposition dans la liste correspondra au nombre de cents que représente cette décomposition. La décomposition qui correspond à zéro cent peut dès à présent être mise dans cette liste car elle est facile.

— Faire une boucle sur tous les montants intermédiaires depuis 1 cent jusqu'au montant final que l'on veut déterminer.

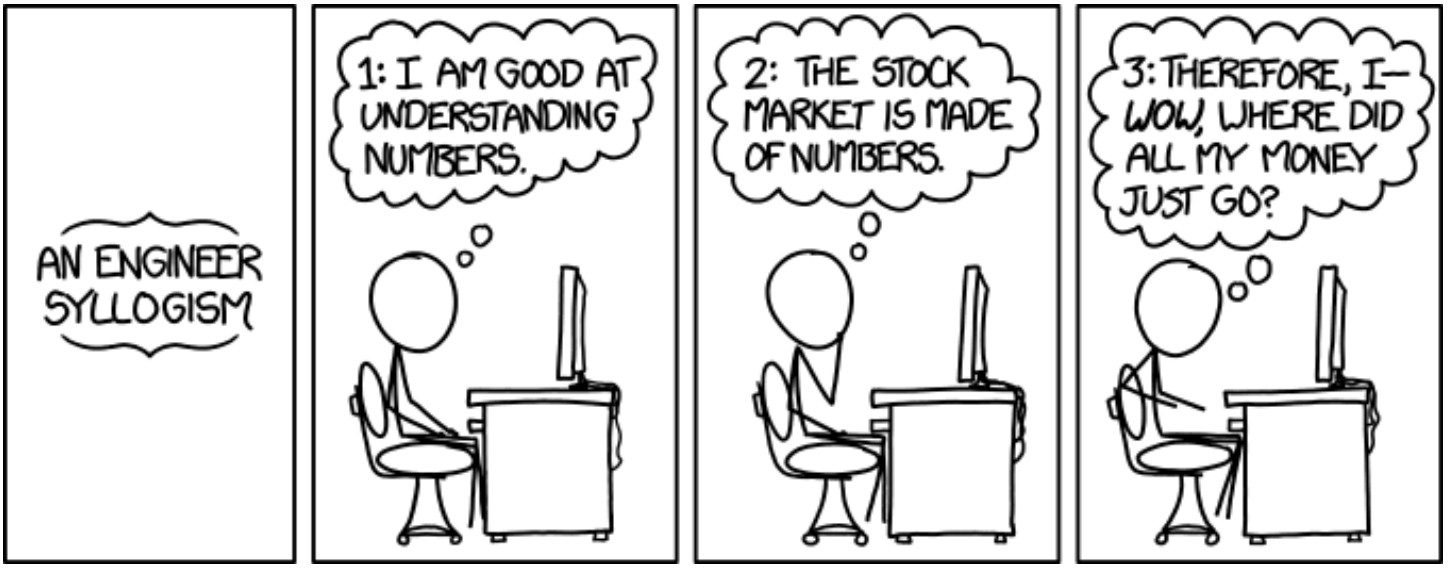
  — Dans cette boucle, initialiser une décomposition qui comporte à l'évidence trop de pièces (par exemple autant de 1 cent qu'il n'y en a dans la valeur à distribuer).

  — Faire une boucle interne à la précédente sur toutes les valeurs faciales de pièces disponibles et se demander de quelle somme partir pour arriver au montant voulu en rajoutant une unique pièce de cette valeur faciale.

  — Cette somme a forcément été déjà rencontrée avant donc on a déjà stocké sa décomposition optimale : la récupérer [8] et regarder si en ajoutant la pièce mise de côté on arrive à une décomposition qui fait intervenir un nombre totale de pièces inférieur.

  — À la sortie de la boucle interne, on stocke la décomposition optimale qu'on vient de trouver.

— Quand on sort de la boucle externe, la dernière décomposition trouvée est celle qu'on attendait.



.com

```
The less common, even worse outcome:
«[everyone in the financial system] WOW, where did all my money just go?»
```

8. En la *copiant* sinon vous allez modifier ce que vous avez déjà stocké. Voir la vidéo dédiée sur la chaîne YouTube