Séance 1
activité débranchée : le crêpier psychorigide

Objectifs :

- Résoudre un problème concret
- Raisonnement algorithmique
- Élaborer une stratégie de résolution de problème complexe en une série de tâches simples
- Discussion autour des solutions proposées : unicité, faisabilité, complexité…

Modalité : groupe de 4 élèves

Matériel : 8 disques de différents diamètres, dont une face est colorée (partie brûlée de la crêpe)

Pré-requis : manipulations des listes, aucun sur les tris

durée : 1h30

Elements de programme

Compétences transversales :

- Analyser et modéliser un problème en termes de flux et de traitements d'informations
- Décomposer un problème en sous-problèmes.
- Concevoir des solutions algorithmiques

Contenus	Capacités attendues
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.
Spécification	Prototyper une fonction. Décrire les préconditions sur les arguments. Décrire des postsconditions sur les résultats.
Mise au point de programme	Utiliser des jeux de tests.

Contexte :
A la fin de sa matinée de travail de travail, un crêpier se rend compte que ses crêpes n'ont pas le même diamètre et que sa pile est totalement désordonnée.. Étant un peu psychorigide, il décide de les ranger de la plus grande à la plus petite en partant du bas de la pile.
Pour effectuer cette tâche, il ne peut exécuter qu'un seul type d'action avec sa spatule : glisser sa spatule entre deux crêpes et retourner le haut de la pile de crêpes.

Lien vidéo (en anglais): https://www.youtube.com/watch?v=m3drS_8BpU0

Activité préparatoire:
Vous avez à votre disposition 8 crêpes de diamètres différents empilées dans un ordre aléatoire et une spatule. Vous devez concevoir une méthode de travail simple à utiliser pour aider le crêpier à trier ses crêpes de la plus grande à la plus petite.

Aides possibles en cas de blocage : où doit se trouver la grande crêpe pour pourvoir l'amener en bas de la pile ? Où doit-on placer la spatule pour retourner la pile ?…

Décrire à l'aide de verbes d'action simple (placer la spatule, retourner…) un algorithme permettant de trier les crêpes à l'aide du seul mode d'action possible de la spatule décrit précédemment.

Algorithme possible :
- trouver la position de la plus grande crêpe,
- mettre la spatule sous la plus grande crêpe et retourner le haut de la pile,
- mettre la spatule tout en bas et retourner l'intégralité des crêpes,
- recommencer la procédure en ignorant les crêpes rangées.

Complément 1 de l'activité :
Un élève parmi vous joue le rôle du crêpier. Un autre élève joue le rôle de l'instructeur.

Contrainte : l'instructeur ne peut que parler. Le crêpier ne peut que exécuter les actions que lui dicte l'instructeur.

Objectif : vérifier que votre algorithme fonctionne et se termine une fois la pile de crêpes triées.

Complément 2 de l'activité :
Même chose que précédemment mais cette fois, avec la contrainte supplémentaire :l'instructeur ne regarde pas la pile de crêpes. La seule chose qu'il connaît, est le nombre de crêpes que contient la pile.

Le crêpier psycho-rigide
cours

Dans l'activité de découverte précédente, une pile désordonnée de n crêpes devait être rangée de la plus grande (en bas) à la plus petite (en haut). Vous avez bâti un algorithme pour résoudre ce problème :

Algorithme possible :

- trouver la position de la plus grande crêpe,
- mettre la spatule sous la plus grande crêpe et retourner le haut de la pile,
- mettre la spatule tout en bas et retourner l'intégralité des crêpes,
- recommencer la procédure en ignorant les crêpes rangées.

Plusieurs questions peuvent se poser.

1- Quel que soit le nombre de crêpes à trier, l'algorithme s'arrête-t-il ?

La terminaison est l'étude d'un algorithme. Elle a pour but de vérifier qu'il se termine au bout d'un certain nombre d'opérations élémentaires. Il ne s'exécute pas indéfiniment. On obtient au moins un résultat. Par contre la terminaison ne garantit pas que le résultat soit juste.

Correction :
« Recommencer cet algorithme » se traduit par une boucle (finie ou infinie). A chaque tour de boucle, le nombre de crêpes à trier diminue d'une unité. Il y a donc n crêpes à trier au premier tour de boucle, puis n-1… jusqu'à 0. L'algorithme s'arrêtera donc à chaque fois quel que soit le nombre de crêpes initiales.

2- L'algorithme a-t-il fait ce que j'attendais de lui ?

La correction garantit que l'algorithme donne le résultat attendu lorsqu'il se termine.

Un algorithme est correct s'il fait ce qu'on attend de lui. Votre algorithme reçoit des données (les positions des crêpes dans une pile désordonnée) sur lesquelles l'algorithme va démarrer son calcul et produire un résultat (les positions dans une pile ordonnée de crêpes).
Pour s'assurer qu'un algorithme est correct, il faut démontrer deux choses: il faut démontrer que l'algorithme se termine (terminaison), produisant au moins un résultat et que le résultat de l'algorithme est effectivement une pile ordonnée de crêpes.
La partie essentielle de votre algorithme est la boucle (tant que ou while) : C'est un algorithme itératif. Pour montrer la correction (partielle) d'un tel algorithme, il faut trouver un invariant pour la boucle. **Un invariant** est une propriété qui est vraie avant et après chaque répétition. Si l'invariant est satisfait avant l'exécution du corps de la boucle, il est satisfait après.

Correction :
L'invariant est « Au bout de k itérations les k premières crêpes sont rangées. ». En effet après un coup on est sûr qu'une crêpe est rangée. Après deux coups, deux crêpes au moins sont rangées...

3- Quelle quantité de ressources l'algorithme a-t-il utilisé ?

L'analyse de **la complexité** d'un algorithme sera traité pendant la séance 3

Lorsque l’on crée un algorithme, il faut donc tenir compte des trois critères suivant : la terminaison, la correction et la complexité.

Séance 2 : programmation de l'algorithme

Objectifs :

- Comprendre un programme
- Effectuer des tests de contrôle d'un programme
- Traduire un algorithme en langage de programmation

Modalité : individuellement

Pré-requis : manipulations des listes, aucun sur les tris

durée : 2h00

Elements de programme

Compétences transversales :

- Traduire un algorithme dans un langage de programmation
- Comprendre et réutiliser des codes sources existants
- Développer des processus de mise au point et de validation de programmes

Contenus	Capacités attendues	commentaire
Parcours séquentiel d'un tableau	Recherche d'un extremum	Montrer que le coût est linéaire.
Tri	Ecrire un algorithme de tri Décrire un invariant de boucle qui prouve la correction	Justifier la terminaison

Activité1 : Analyse d'une fonction

Soit le programme en python suivant :

```
def mystereA(tab, index):
    k = 0
    for i in range(index+1):
        if tab[i] > tab[k]:
            k = i
    return k

tab = [1, 12, 5, 6, 13, 8, 14, 7, 9, 3]
print(mystere(tab,len(tab))
```

Qu'affiche la console de python après l'exécution du programme ? 6

Si on on remplace la dernière ligne par : print(mystere(tab,4), qu'affiche la console de python après l'exécution du programme ? 1

Renommer la fonction à l'aide d'un nom explicite : par exemple, indice_du_plus_grand_nombre()

Pour revenir sur le cours : notion d'invariant à aborder dans cette fonction (k), le programme se finit-il ?

Activité2 : Ouvrir le programme python « mystereB.py »
Ajouter une ligne au programme pour qu'il affiche en console le résultat de

- mystereB(Tab,len(tab))

Rep : (10, 9, 8, 7, 6, 5,4,3, 2, 1)

- mystereB(Tab,4)

Rep : (5,4,3, 2, 1, 6, 7, 8,9 10)

Que fait cette fonction ? Elle écrit des k-1 premiers éléments de la liste à l'envers. Le reste n'est pas modifié.

Renommer la fonction à l'aide d'un nom explicite. Par exemple, rotation_liste()

Activité 3 : ouvrir le programme « crepes-eleves .py »
Le programme correspond à l'algorithme de tri des crêpes étudié précédemment.
Les crêpes sont représentées par des entiers positifs dont la valeur correspond au rayon de la crêpe.
On remarque que les entiers (crêpes) ne sont pas rangées dans la liste.

Travail à faire :

- Reconnaître les 2 fonctions déjà étudiées précédemment.
- Vous devez compléter les lignes 37, 39 et 42 afin de trier les crêpes.

Pour aller plus loin :

Lien video : https://www.youtube.com/watch?v=o6-4g4I6mOg

Les crêpes présentent une face brûlée, l'autre non. Elles sont empilées initialement de façon désordonnée. Le crêpier doit les ranger de la plus grande à la plus petite mais avec la face brûlée toujours en dirigé vers le bas.
Pour modéliser les crêpes présentant une face brûlée et l'autre non, on utilise un tableau de la forme :

pile = [1, -4, 5, 2, -12, 8, 6, -7, 9, -3]

Un entier négatif correspond à une crêpe avec le côté brûlé vers le haut.
Un entier positif correspond à une crêpe avec le côté brûlé vers le bas.
La valeur absolue de l'entier correspond toujours au rayon de la crêpe.

Travail à faire:

- Utiliser le modèles des crêpes pour écrire l'algorithme correspondant
- Avant de programmer, que devrait afficher le programme en utilisant la liste « pile » précédente ?
- Copier avant de commencer à programmer, le programme « « crepes-eleves .py » et le renommer « crepes_brulees-eleves .py »
- Modifier le programme utilisé « crepes_brulees-eleves .py » pour répondre à l'objectif.

Aide :
- quand la plus grande crêpe est en haut de la pile, il faut vérifier que la face du dessus est brûlée, sinon il faut la retourner.
- quand on retourne le haut de la pile de crêpes, toutes les faces des crêpes sont inversées. Dans ce cas, il faut inverser le signer des entiers de tous les éléments de la liste retournée.

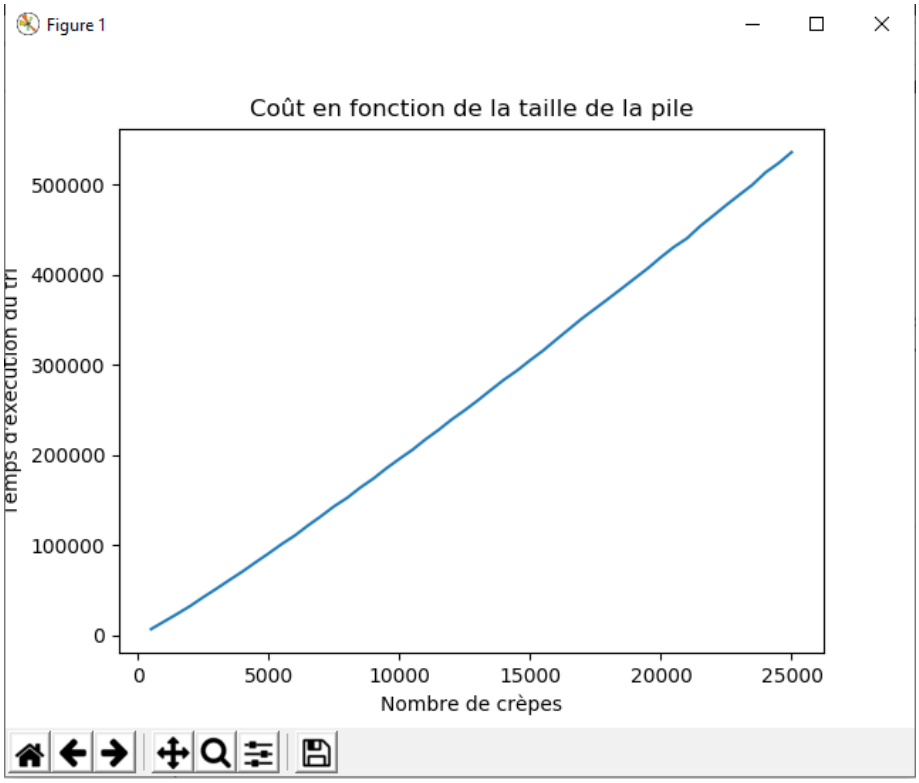
Séance 3 : quelle quantité de ressources l'algorithme a-t-il utilisé ?

L'analyse de **la complexité** d'un algorithme consiste en l'étude de la quantité de ressources (par exemple de temps ou d'espace) nécessaire à l'exécution de cet algorithme. Il est possible de calculer la quantité de ressources (le temps de calcul s'il s'agit d'un ordinateur) dans le pire des cas, le meilleur des cas ou la moyenne des cas. Mais le plus intéressant est la quantité de ressources utilisées dans le pire des cas. C'est une notion qui sera étudiée en terminale.
Pour évaluer la performance, on compte le nombre de «coups» nécessaires pour résoudre le problème dans le cas général. Pour le problème du crêpier:

En pseudo-code :
• pour ranger une crêpe, il faut entre 0 coup (la crêpe est déjà rangée) et 3 coups (amener en haut, retourner, amener à sa place) ;
• pour n crêpes (cas général), il faut entre 0 coup (meilleure situation) et 3×n coups (pire situation).

Première idée : compter les opérations
Reprendre l'algorithme et réfléchir à la manière d'insérer des compteurs sur le nombre d'opérations.
Note : Pas si simple… Il y a plusieurs fonctions, il faut utiliser deux compteurs « intermédiaires ».

A l'aide du fichier « crepe_psychorigide-coutv3.py », on peut visualiser une simulation sur différentes tailles de piles (entre 500 et 25 000)



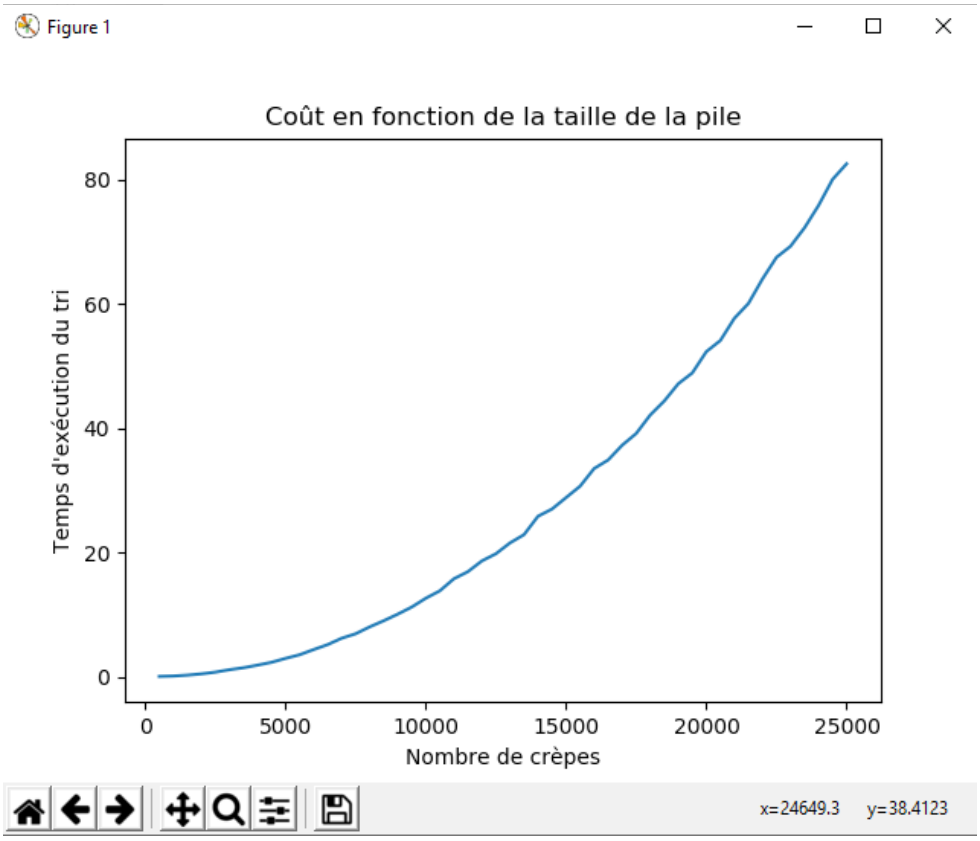
Le graphique semble être la représentation d'une fonction linéaire : f(n) ≈ 20n.

Autrement dit, **le temps de calcul croît de façon linéaire en fonction du nombre de crêpes n**. Si on traite deux fois plus de données, le temps d'exécution sera multiplié par deux.
On parle alors d'un algorithme de **complexité linéaire** et on note **O(n)**.

Mais tous les coups ne demandent pas la même quantité de ressources. Il faut plus d'énergie pour retourner 10 crêpes que pour en retourner 5.
Le temps d'exécution variant selon l'état initial, la performance exprime l'ordre de grandeur du temps d'exécution.

Seconde idée : Mesurer le temps écoulé
Reprendre l'algorithme et réfléchir à la manière d'insérer une mesure du temps d'exécution.
On utilise la bibliothèque *time* et les instructions :
start_time = time.time() au début
ainsi que *temps = time.time() - start_time à la fin du programme principal*

A l'aide du fichier « crepe_Cout_Graphique.py », on peut visualiser une simulation sur différentes tailles de piles (entre 500 et 25 000)



Le graphique n'est plus linéaire. Il semble être la représentation d'une fonction carré : $g(x) \approx C*n^2$. (C constante)
Un algorithme avec une performance **$O(n^2)$** est dit de **complexité quadratique.**
Questionnement :

On ne trouve pas le même type de complexité en regardant le nombre d'opérations qu'en regardant le temps. Comment l'expliquer ?

Correction :
On a utilisé la fonction *pile_modifie.reverse()*
Or cette fonction coûte plus qu'une opération. Elle cache une boucle.

Exercice1 :
Comment inverser les éléments d'une liste ?
Ouvrir un nouveau fichier Python et définir une fonction rotation_liste ayant deux arguments en entrée : pile et index

Correction :
```
def rotation_liste(pile, index):
    nombre_inversions = index // 2
    for i in range(nombre_inversions):
        j = index - 1 - i
        aux = pile[i]
        pile[i] = pile[j]
        pile[j] = aux
    return pile
```

Exercice 2 :
comme précédemment, reprendre l'algorithme et réfléchir à la manière d'insérer un compteur sur le nombre d'opérations.

Correction :
On effectue 4 opérations par boucle. On remarque que la variable i nous permet de compter le nombre de boucles effectuées.
La variable « compteur » s'incrémente de 4 à chaque itération.
Le nombre d'inversion étant égal à index//2, au pire cas on n//2 inversions.
Autrement dit la **complexité est linéaire** pour cette boucle.

Synthèse :
La complexité globale de l'algorithme est bien en $O(n^2)$ donc quadratique.

Correction :
Si on considère seulement le nombre de coups, la performance de l'algorithme est proportionnelle au nombre de coups. Il est donc linéaire. Attention ! Cependant les coups ne sont pas des opérations élémentaires pour l'ordinateur. Le temps de calcul augmente de façon quadratique avec la taille de la pile de crêpes.

Lorsque l'on crée un algorithme, il faut donc tenir compte des trois critères suivant : la terminaison, la correction et la complexité.