R1.01 Initiation au développement
Feuille de TD n°14
Algorithmes Gloutons

Objectifs de la feuille

- On continue à travailler la représentation de la mémoire (oui, encore !)
- Et on continue à travailler l'algorithmique avec quelques algorithmes gloutons.

Exercice 1 *Représentation de la mémoire*

1.1 On exécute le script suivant. Représenter l'état de la mémoire au premier passage par l'endroit indiqué `# ICI` .

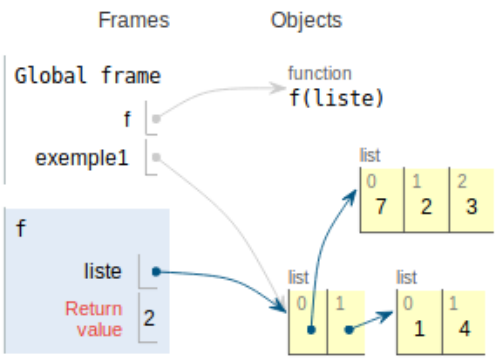
1.2 Préciser la valeur de `dernier` à l'issue du script.

```
def reste(nombre):
  exemple = nombre % 3
  # ICI
  return exemple

def truc(liste):
   somme = 0
   for elem in liste:
       somme = somme + reste(elem)
   return somme

def ajoute(liste):
    copie = liste.copy()
    copie.append(7)
    return copie

exemple = [10, 5, 9]
suivant = ajoute(exemple)
dernier = truc(suivant)
```

Exercice 2 *Représentation de la mémoire*

2.1 Compléter le script donné sachant qu'il provoque la représentation de la mémoire proposée. Préciser à quel endroit dans le script on se trouve.

```
# Script à compléter
exemple1 = [[7, 2], [1, 4]]
```



Exercice 3 *Min, max et sorted*

3.1 Quelle est la valeur de `mystere1` et `mystere2` après l'exécution du code suivant ?

```
def f1(couple):
    return couple[1]

def f2(couple):
    return couple[0]%10

exemple = [(16, 6), (12, 3), (9, 1), (42, 3), (75, 7), (27, 6)]
mystere1 = min(exemple, key = f1)
mystere2 = sorted(exemple, key = f2)
```

3.2 Proposer un code de la fonction `plus_grand` :

```
def plus_grand(liste):
    """
    parametre : une liste de couples de nombres (non vide)
    resultat : le couple dont la somme des composantes est la plus grande
    """
    ...
```

```
assert plus_grand([(16, 6), (9, 1), (2, 3), (75, 7), (27, 6)]) == (75, 7)
```

3.3 Proposer un code de la fonction `le_moins_de_voyelles` :

```
def le_moins_de_voyelles(liste):
    """
    parametre : une liste de str non vide
    résultat : le str de la liste qui possède le moins de voyelles
    """
    ...
```

```
assert le_moins_de_voyelles(['je', 'suis', 'ton', 'pere']) == 'je'
assert le_moins_de_voyelles(['you', 'know', 'nothing']) == 'know'
```

3.4 Proposer un code de la fonction `tri_selon_nombre_de_villes` :

```
def tri_selon_nombre_de_villes(region):
    """
    region est un dictionnaire dont les clés sont des régions et
    la valeur associée un ensemble de villes
    renvoie la liste des clés rangées dans l'ordre croissant de leur
    nombre de villes
    """
    ...
```

```
centre = { \
    'Loiret' : {'Orleans', 'Trainou', 'Chécy', 'Ardon', 'Amilly'},
    'Cher' : {'Vierzon', 'Quincy', 'Bourges'},
    'Indre' : {'Chateauroux', 'Issoudin', 'Déols', 'Le Blanc'}}
assert tri_selon_nombre_de_villes(centre) == ['Cher', 'Indre', 'Loiret']
bretagne = { \
    'Morbihan': {'Lorient', 'Vannes', 'Ploemeur'},
    'Finistère' : {'Cast', 'Brest', 'Quimper', 'Concarneau'},
    'Cotes Armor': {'Lannion'}}
assert tri_selon_nombre_de_villes(bretagne) == \
    ['Cotes Armor', 'Morbihan', 'Finistère']
```

Exercice 4 *Lecture de code*

On représente un intervalle `intervalle = [debut, fin[` par le tuple `(debut, fin)` .

```
def mystere(intervalle1, intervalle2):
    """
    paramètres : deux intervalles [debut, fin[ représentés par des tuples
    """
    (debut1, fin1) = intervalle1
    (debut2, fin2) = intervalle2
    return debut1 < fin2 and debut2 < fin1
```

4.1 Préciser le type et la valeur des variables `a` , `b` , `c` et `d` :

```
a = mystere((1, 4), (6, 8))
b = mystere((7, 9), (2, 4))
c = mystere((1, 11), (6, 8))
d = mystere((8, 14), (5, 10))
```

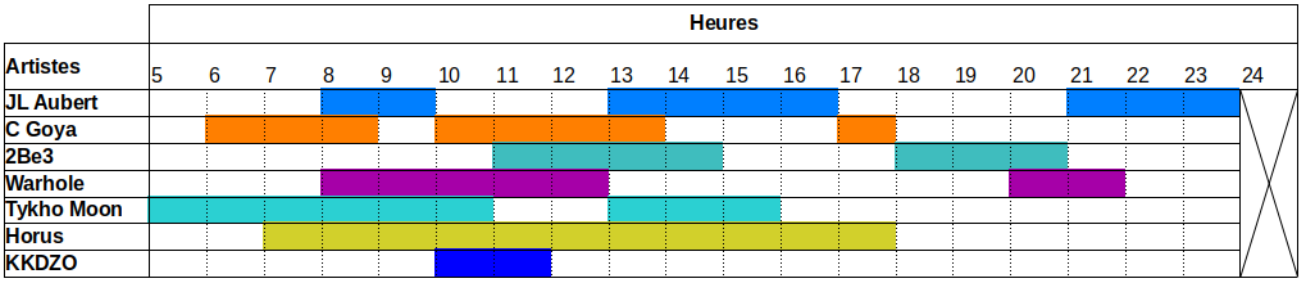
4.2 En une ou deux phrase, explique ce que fait la fonction `mystere`

Exercice 5 *Planning pour un festival : algorithmique sans code*

Au cours d'un festival, plusieurs spectacles ont lieu, souvent sur plusieurs scènes. On veut écrire une application pour aider des festivaliers à assister à un maximum de spectacles.

Par exemple, le Festival *Enki Bilal* se déroule tous les ans, le 32 décembre, à Paris. Ce festival commence tôt le matin (dès 5h) et se termine à minuit. Les artistes à l'affiche cette année sont les suivants :

- JL Aubert se produira de 8h à 10h. Après une courte pause, il donnera un deuxième spectacle de 13h à 17h. Puis, il reviendra plus tard dans la soirée de 21h à 24h.
- La grande artiste C Goya donnera également trois représentations. La première de 6h à 9h, la deuxième de 10h à 14h et la dernière de 17h à 18h.
- Les 2Be3 viendront nous honorer de leur présence. Ils donneront deux spectacles : l'un de 11h à 15h, et l'autre de 18h à 21h.
- Le groupe de musique local Warhole se produira de 8h à 13h puis de 20h à 22h.
- Le célèbre Tykho Moon donnera deux concerts : le premier de 5h à 11h et le second de 13h à 16h.
- La troupe de théâtre Horus assurera un spectacle continu entre 7h et 18h.
- Enfin, le groupe KKDZO assurera le show de 10h à 12h.



5.1 On considère que les spectacles ont lieu pendant l'intervalle de temps `[début, fin[` et on considère que deux spectacles sont compatibles si leur deux intervalles de temps ne se superposent pas. Exhiber deux spectacles compatibles et deux spectacles incompatibles.

5.2 Anakin, grand fan du festival Enki Bilal, veut assister au maximum de spectacles possibles. Il prévoit d'arriver dès le début du festival à 5h. Il a fait la pré-sélection suivante : Tykho Moon à 5h, JL Aubert à 13h et les 2Be3 à 18h. Il se demande s'il n'est pas possible d'aller voir plus de spectacles. Pouvez-vous lui proposer une sélection de spectacles plus fournie que la sienne ?

5.3 Quelle est, d'après-vous, la sélection de spectacles optimale, c'est-à-dire celle qui comporte le plus grand nombre de spectacles ? (Pas de preuve demandée ici, juste une solution intuitive).

5.4 Un premier algorithme

Pour maximiser le nombre de spectacles vus, Anakin décide d'utiliser l'algorithme suivant :

```
Données :
  * programme : tous les spectacles du festival avec leur heure de
                debut et leur heure de fin
Résultat : les spectacles sélectionnés (tous compatibles)
Algorithme :
  * initialiser selection à la liste vide
  * initialiser heure_actuelle à l'heure de début du festival
  * initialiser selection_est_en_cours à True
  * tant que la selection_est_en_cours
      - on choisit dans le programme le premier spectacle qui commence
        après heure_actuelle. Puis :
         + on ajoute ce spectacle à la sélection
         + heure_actuelle prend la valeur de l'heure de fin du spectacle
      - si aucun spectacle ne peut être choisi, selection_est_en_cours
        prend la valeur False.
  * Renvoyer la selection
```

S'il arrive dès le début du festival, quelle sélection lui proposera cet algorithme ?

5.5 Un deuxième algorithme

Pour maximiser le nombre de spectacles vus, Anakin a une autre idée : il décide de favoriser les spectacles de petite durée. Proposer un algorithme qui utilise la durée des spectacles comme critère de choix.

Avec cet algorithme, s'il arrive dès le début du festival, quelle sera sa sélection ?

5.6 Un troisième algorithme

Anakin a une autre idée : il choisit tour à tour le spectacle compatible avec sa sélection qui se termine le plus tôt possible possible.

Avec cet algorithme, s'il arrive dès le début du festival, quelle sera sa sélection ?

5.7 Réflexion sur un début d'implémentation

a) Quelle structure de données pourrait représenter un `spectacle` ? Donner un exemple de modélisation du premier spectacle de JL Aubert.

b) Quelle structure de données pourrait représenter le programme complet d'un festival ? Donner un exemple de modélisation du programme du festival de Nikopol.

c) Quelle structure de données pourrait représenter une sélection de spectacles ? Donner un exemple avec la sélection de Anakin.

La suite de cette feuille se trouve sur CELENE