TP 6| Algorithmes gloutons

Le principe des algorithmes gloutons consiste à essayer de répondre à une problématique en choisissant, à chaque étape, la meilleure solution « locale », en espérant que la solution finale sera la meilleure des solutions globales.

1. Rendu de monnaie (1h)

On dispose d'un ensemble de pièces et billets en euros :

On souhaite faire de la monnaie sur une somme donnée avec le moins de pièces/billets possible.
Le principe de l'algorithme glouton dans le cas de rendu de monnaie consiste à rendre la monnaie en tentant de rendre à chaque fois, le plus grand billet ou la plus grande pièce possible, puis de répéter l'opération jusqu'à ce qu'il ne reste plus rien.
On suppose l'existence d'une variable globale « Système », pointant vers une liste de toutes les valeurs possibles de pièces et billets, ordonnées de la plus petite à la plus grande. Par exemple, Système=[1,2,5,10,20,50,100,200,500] en euros (on peut aussi ajouter les centimes).

1) Créer une fonction `Piece_Billet(Reste)` qui prend en argument une valeur à rendre (un nombre), et qui renvoie la plus grande valeur possible à rendre dans le système utilisé. Par exemple, dans notre système monétaire en euro, `Piece_Billet(72)` doit renvoyer 50.

2) Créer une fonction `Monnaie(Somme)` qui crée et renvoie la liste du rendu de monnaie associé à la somme entrée, en respectant le principe de l'algorithme glouton, c'est-à-dire la liste des valeur consistant à « faire la monnaie » sur la valeur `Somme.` On commencera arrondira au centième.

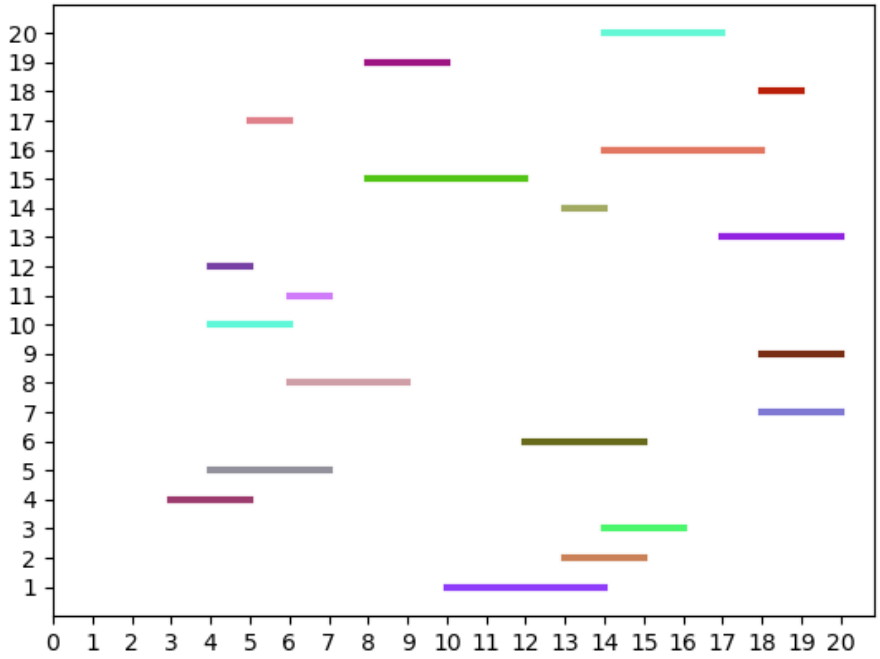
3) Mettre en place le code permettant de déterminer et afficher dans la console la monnaie de 77€22.

Dans le système en euros, nous admettrons que cet algorithme est optimal, on dit qu'il est « canonique ».

4) Montrer par un exemple, qu'il existe des systèmes monétaires pour lesquels l'algorithme glouton proposé n'est pas optimal. On pourra s'inspirer du système monétaire [4,3,1].

2. Allocation d'une ressource (1h30)

On se place dans le cadre de la réservation d'une salle (ressource).
Nous avons à disposition plusieurs demandes de réservations R_i, qui précisent donc une date de début D_i et une date de fin F_i. Voici un exemple de 20 demandes (axe des ordonnées) réparties sur 20 jours (axe des abscisses) avec des durées différentes :



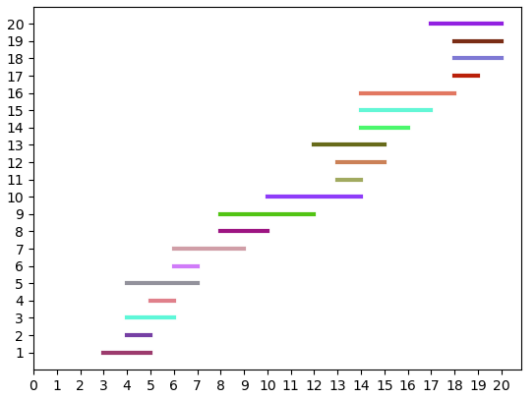
Les conditions à respecter sont les suivantes :

- On ne peut pas donner la salle à deux personnes le même jour ;
- L'objectif de l'optimisation est de répondre au plus grand nombre de demandes.

Remarque : Le plus de réservation n'est pas forcément compatible avec la plus grande occupation de la salle. Par exemple, 3 demandes consécutives d'un jour (sur 3 jours au total) seront privilégiées face à une demande de 4 jours…

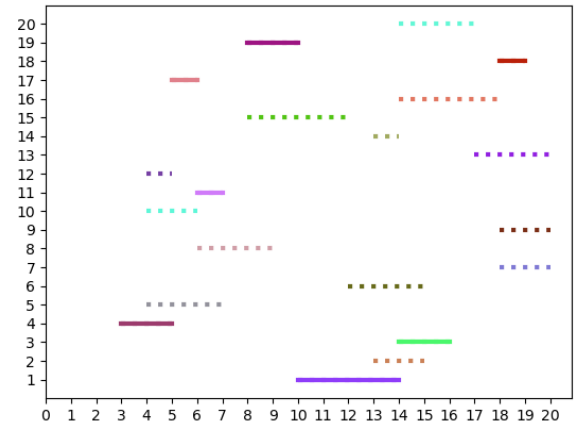
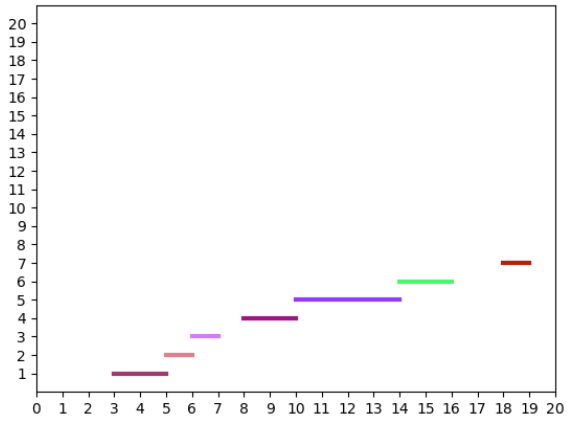
Le principe de l'algorithme glouton dans ce cas consiste à :

- Trier les demandes par ordre croissant de fin :



- Parcourir alors les demandes triées en retenant à chaque fois la réservation compatible ($D_i \geq F_{i-1}$) la plus courte, pour en mettre bout à bout le plus possible. C'est donc une solution optimale locale qui est recherchée à chaque itération.

On trouve alors la solution suivante :

Ensemble des demandes en pointillés, celles retenues en traits pleins, selon l'ordre des demandes initiales	Ensemble organisé des demandes acceptées, soit le planning d'occupation de la salle
	

Génération des réservations

On souhaite générer une liste de réservations aléatoires comprises entre début (numéro du premier jour inclus de type entier) et Fin (numéro du dernier jour inclus de type entier) de durée max Duree_Max de type entier et telles que chaque réservation soit une liste $[D_i, F_i, i, Couleur]$ avec :

- $D_i \geq Debut$
- $F_i \leq Fin$
- $1 \leq F_i - D_i \leq Duree\_Max$
- i est le numéro de réservation, la première étant d'indice $i = 1$
- $Couleur$ est une liste de 3 nombres aléatoires dans $[0,1]$ permettant de définir une couleur RGB

Pour rappel, on obtient dans le module random :

- Un entier aléatoire dans l'intervalle [a,b] avec randint(a,b).
- Un réel aléatoire dans l'intervalle [a,b] avec uniform(0,1).

1) Proposer une fonction Generation(Debut,Fin,Duree_max,Nb) permettant de générer une liste de Nb sous-listes de type [D,F,Indice,Couleur].

Vérifiez quelques exemples :

```
>>> Generation(0,5,2,5)
[[3, 4, 1, [0.4843192151615079, 0.20766460848757173, 0.5901621536654499]],
[2, 3, 2, [0.34734071482631157, 0.6904222913788538, 0.10852260069347253]],
[3, 5, 3, [0.6914586120001831, 0.06256982006673195, 0.99083822066321 46]], [
3, 4, 4, [0.37832039769517056, 0.8142201076732741, 0.62184906658167 07]], [0
, 1, 5, [0.5798197498377815, 0.5876176730550857, 0.99565675960692 65]]]

>>> Generation(0,2,2,5)
[[1, 2, 1, [0.0672254354770836, 0.9177783743147795, 0.35596933555049 35]], [
1, 2, 2, [0.562221995856825, 0.9541314855886106, 0.69769735470722 8]], [1, 2
, 3, [0.30342223887113795, 0.45732225104455637, 0.14266380040744753]], [1,
2, 4, [0.7063868722349047, 0.09027108419734131, 0.48607035383709607]], [1,
2, 5, [0.1131446125561788, 0.31562839439704304, 0.60225610454908 23]]]

>>> Generation(10,20,3,5)
[[10, 11, 1, [0.15556512978159476, 0.40210214049195714, 0.81756212696262 49]
], [15, 17, 2, [0.050345108117127224, 0.08175886969658375, 0.73648073493824
22]], [16, 19, 3, [0.7741329800741917, 0.727747682301916, 0.43897074785481 95]], [17, 19, 4, [0.29361958813498235, 0.7692899280929002, 0.42262797099696
237]], [10, 12, 5, [0.010285499265123343, 0.5576230869902049, 0.63707442924
21786]]]
```

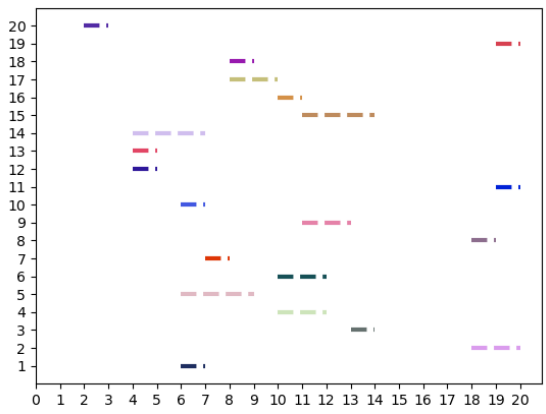
Affichage

On souhaite créer une fonction Affiche(fig,LR,Type,Ordre) qui trace l'ensemble des réservations de la liste LR en argument sur la figure numéro « fig » avec la couleur associée (triplet contenu dans les réservations), avec les options suivantes :

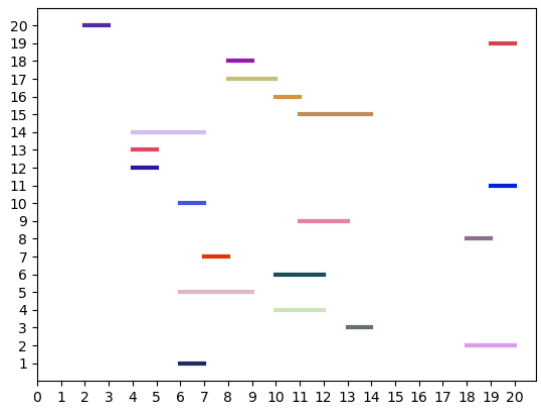
- Type : Chaine de caractères définissant le type de trait tracé
  - Pointillés : '--'
  - Trait plein : '-'
- Ordre : Les réservations sont placées sur l'axe des ordonnées selon :
  - S'il vaut 1, leur ordre d'apparition dans la liste LR
  - S'il vaut 0, leur indice ind

```
Res = Generation(0,20,3,20)

Affiche(1,Res,"--",1)
```



```
Affiche(1,Res,"-",0)
```



On précise que trois variables globales seront définies par la suite et seront utilisées dans la fonction Affiche afin de créer les graduations sur les axes :

- Nb : Nombre de réservations (Entier) – Avec « yticks », on attend une graduation par réservation. La commande plt.yticks(L) affiche des graduations associées aux valeurs contenues dans L.
- Debut et Fin : Premier et dernier jour de toutes les réservations (Entiers) – Avec « xticks », on attend une graduation par jour entre début et fin

2) Créer la fonction Affiche comme demandé et vérifier qu'elle fonctionne. On rappelle que la commande plot du module pyplot, avec la commande plot(X,Y,Type,color=col), permet de relier les points d'abscisses X, d'ordonnées Y, avec le type donné par Type et de couleur col.

Essayez :

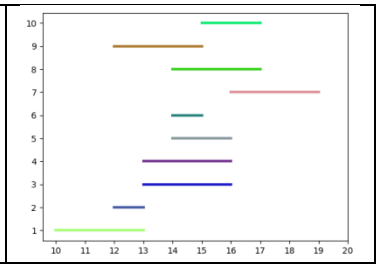
```
Debut = 10

Fin = 20

Nb = 10

Res = Generation(Debut,Fin,3,Nb)

Affiche(0,Res,"-",0)
```



A ce stade, on obtient évidemment le même tracé que l'on demande Ordre=0 ou 1.

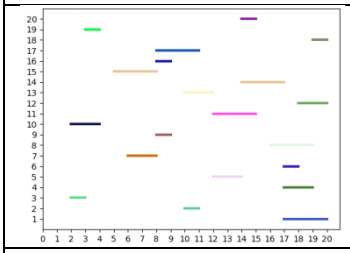
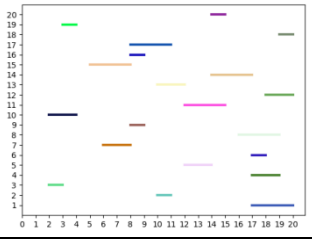
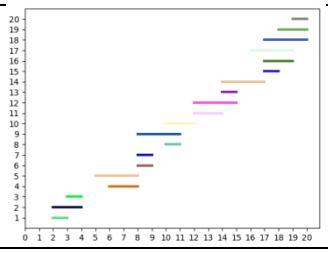
Mise en place du tri

3) Créer la fonction Tri_fin(LR) qui permet de trier les réservations de la liste de réservations LR selon leur fin. On pourra utiliser la commande L.sort() qui permet de trier une liste L, en ayant en tête que derrière se cache un algo compliqué.

Vous pourrez alors vérifier que vous obtenez des images similaires à celles-ci :

```
Res = Generation(0,20,3,20)

Res_Tri = Tri_fin(Res)
```

`Affiche(0,Res,"-",0)` `Affiche(0,Res,"-",1)`	`Affiche(1,Res_Tri,"-",0)`	`Affiche(2,Res_Tri,"-",1)`
		
Les réservations ont beau être triées, si on décide de les afficher selon leurs indices dans Res_Tri (à droite), on retrouve leur ordre de création dans Res (à gauche)		Si on affiche les réservations par ordre d'apparition dans Res_Tri, elles sont bien triées par ordre de fin

Résolution par l'algorithme Glouton

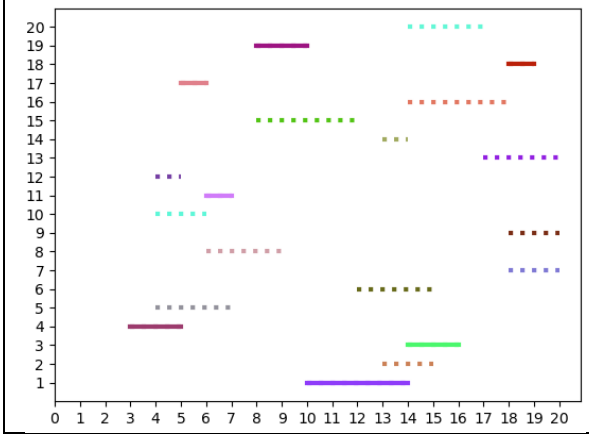
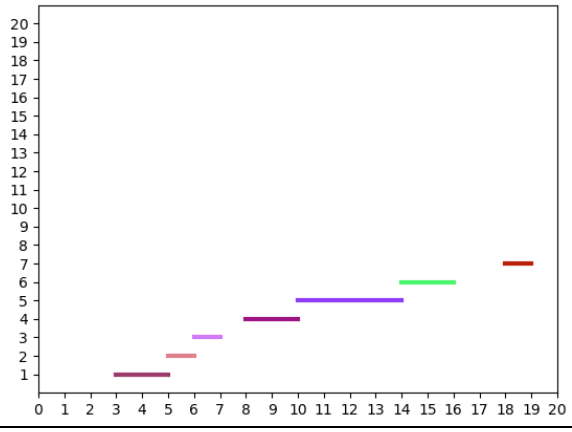
4) Mettre en place une fonction Resolution(LR) prenant une liste de réservations quelconque LR, et renvoyant la liste des réservations choisies par l'algorithme Glouton décrit précédemment.

Application

On définira les variables globales suivantes :

```
Debut = 0
Fin = 20
Duree_max = 4
Nb = 20
```

5) Mettre en place le code permettant, à partir de ces données, de générer les réservations, de déterminer par l'algorithme glouton la solution de distribution des réservations, et d'afficher sur deux figures différentes :

L'ensemble des réservations en pointillés, et par-dessus, les réservations choisies par l'algorithme en trait continu	Le planning d'occupation de la salle
	

Remarque : Compte tenu de notre programmation, l'affichage du planning de la salle indique Nb réservations alors qu'il y a moins de réservations retenues. Mais, si on définit dans la fonction Affiche, le nombre de réservations comme len(LR), c'est l'image de gauche qui présentera un souci puisqu'en ajoutant les réservations choisies par l'algorithme, l'axe des ordonnées sera redéfini… A vous de choisir !