OBJECTIFS : L'objectif de ce Tp sont de découvrir une situation dans laquelle un algorithme de type Glouton est utilisée. Il est aussi de coder en utilisant le paradigme de programmation objet.

1. Fonction donnant le détail du rendu de pièces et billets :

Une caisse automatique doit rendre une somme d'argent de r euros en utilisant des pièces ou billets dont les valeurs, triées par ordre croissant, sont contenues dans la liste ℓ.
La fonction *glouton*(ℓ,r) permet de constituer et retourner une chaine de caractère donnant les différents billets et pièces à rendre.

Par exemple, l'exécution de

```
r = glouton([1,2,5,10,20,50],88)
print(r)
```

Donne dans le shell :

```
50  20  10  5  2  1
```

⇨ Coder cette fonction en programmation fonctionnelle classique

2. Rendu de monnaie sur un distributeur alimentaire :

Le distributeur ci-contre accepte les pièces de 1ct, 2cts, 5cts, 10cts, 20cts, 50cts, 1€, 2€ et les billets de 5€, 10€, 20€ et 50€.

Au chargement de la machine, elle contient 30 pièces pour chacune des valeurs en pièces et 5 billets pour chacune des valeurs en billets.

2.1. Class Monnaie :

La classe Monnaie permet, pour une valeur monétaire donnée de ce distributeur, d'encapsuler des renseignements utilises.

```
class Monnaie :
    def __init__(self,valeur,nombre=0) :
        self.valeur = valeur
        self.nombre = nombre




    def __str__(self) :
        return (f"{self.format} de  {self.nom} : {self.nombre} {self.format} ")
```

⇨ Coder le constructeur de cette classe afin de pouvoir obtenir à l'exécution des lignes suivantes :

```
p_1ct = Monnaie(0.01,30)
p_1euro = Monnaie(1,30)
b_50euros = Monnaie(50,5)
print(p_1ct)
print(p_1euro)
print(b_50euros)
```

La réponse suivante dans le shell :

```
pièces de  1.0 centimes : 30 pièces
pièces de  1 € : 30 pièces
billets de  50 € : 5 billets
```

2.2. Class Distributeur :

La classe Distributeur permet d'encapsuler les données par rapport à la réserve d'argent contenues dans la machine et des méthodes pour gérer cette réserve et gérer le rendu de monnaie.

```
class Distributeur:
    def __init__(self) :
        self.reserve = []
```

Le constructeur de cette classe permet d'initialiser une liste vide.

⇨ Définir les méthodes *remplirReserve()* et *__str__()* qui permettent à l'exécution des lignes :

```
branly = Distributeur()
for val in [0.01,0.02,0.05,0.1,0.2,0.5,1,2] :
    branly.remplirReserve(val,30)
for val in [5,10,20,50] :
    branly.remplirReserve(val,5)
print(branly)
```

D'obtenir dans le shell :

```
pièces de  1.0 centimes : 30 pièces
pièces de  2.0 centimes : 30 pièces
pièces de  5.0 centimes : 30 pièces
pièces de  10.0 centimes : 30 pièces
pièces de  20.0 centimes : 30 pièces
pièces de  50.0 centimes : 30 pièces
pièces de  1 € : 30 pièces
pièces de  2 € : 30 pièces
pièces de  5 € : 5 pièces
billets de  10 € : 5 billets
billets de  20 € : 5 billets
billets de  50 € : 5 billets
```

⇨ Définir la méthode *payer()* qui permet ensuite, à l'exécution des lignes :

```
branly.payer(6.7,10)
print(branly)
```

D'obtenir dans le shell :

```
à rendre :  2   1   0.2   0.1

pièces de  1.0 centimes : 30 pièces
pièces de  2.0 centimes : 30 pièces
pièces de  5.0 centimes : 30 pièces
pièces de  10.0 centimes : 29 pièces
pièces de  20.0 centimes : 29 pièces
pièces de  50.0 centimes : 30 pièces
pièces de  1 € : 29 pièces
pièces de  2 € : 29 pièces
pièces de  5 € : 5 pièces
billets de  10 € : 6 billets
billets de  20 € : 5 billets
billets de  50 € : 5 billets
```