But

Le but est de revoir la programmation `python`, au travers d'un code défensif mettant en œuvre un algorithme glouton.

La problématique

Choisir 5 villes françaises et chercher sur internet la distance et le prix du transport [www.viamichelin.fr]. Dresser le tableau des prix (essence + autoroute). Après avoir arrondi les prix à l'euro supérieur, on cherche la ville de départ qui permettra un cycle passant par les cinq villes une et une seule fois et revenant à la ville initiale de sorte que le prix majoré du cycle soit le moins élevé.

On donnera alors le cycle sélectionné, le prix ainsi que la décomposition utilisant le moins de devises afin de payer ce prix (coupures de 1, 2, 5, 10, 20, 50, 100, 200 ou 500 euro).
On pensera à faire de la programmation défensive (en vérifiant notamment que les prix arrondis au supérieur sont bien des entiers).

Exemple / aide

Si on choisit les trois villes suivantes Brest, Marseille et Strasbourg.

Nous aurons

	Brest	**Marseille**
Strasbourg	156,77€	121,74€
Marseille	183,47€	

Le cycle demandant le moins de devises coûte au plus 463€, puisque $463 = 2 \times 200 + 50 + 10 + 2 + 1$, le prix nécessite 6 devises.

Étapes

a) Choisir un binôme et quatre villes françaises. Dresser le tableau des prix exacts puis les arrondir à l'euro supérieur. Calculer les prix des trois chemins possibles à partir de la première ville (dans l'ordre alphabétique) et leur décomposition en coupures de 1, 2, 5, 10, 20, 50, 100, 200 ou 500 euro.

Quel est le cycle le moins cher selon un algorithme glouton et quel est le nombre de devises minimal pour payer ?

Par la suite, vous penserez à tester vos différentes fonctions (votre exemple à 4 villes pourra servir de base de tests).

b) Proposer une fonction `decompDevises( prix : int ) -> dict` qui décompose un prix entier positif en un minimum de coupures de 1, 2, 5, 10, 20, 50, 100, 200 ou 500 euro.

Si le prix est négatif ou flottant, on levera une exception respectivement `ValueError` ou `TypeError`.

La décomposition sera contenu dans un dictionnaire associant valeur des coupures à leur nombre nécessaire dans la décomposition.

e.g. `decompDevises( -47 )` lèvera l'exception `ValueError` avec un message d'erreur.

e.g. `decompDevises( 47 )` renverra `{500:0, 200:0, 100:0, 50:0, 20:2, 10:0, 5:1, 2:1, 1:0}`.

c) Proposer une fonction `strDevises( prix : int ) -> str` qui à partir d'un prix entier positif renvoie une chaîne de caractères préciant la décomposition du prix en un nombre minimal de coupures.

e.g. `strDevises( 47 )` renverra la chaîne "2 :20€ ; 1 :5€ ; 1 :2€"

d) Créer une variable `VILLES` qui contient sous forme de tableau le nom des villes dans le désordre. Écrire une fonction `dicoVilles( tab : list ) -> dict` qui à partir d'une liste de noms de villes, trie en place cette liste par ordre alphabétique et créer un dictionnaire associant le nom de la ville à son rang.

e.g. `VILLES = ['Brest', 'Strasbourg', 'Marseille']`, après l'appel de la fonction, `VILLES = ['Brest', 'Marseille', 'Strasbourg']` et renverra `{'Brest':0, 'Marseille':1, 'Strasbourg':2}`

e) Proposer une fonction `verifVilles( n : int, dico : dict ) -> bool` qui renvoie `True` si les valeurs du dictionnaire sont des entiers compris entre 0 et $n-1$, sinon lève une exception, selon le cas `TypeError` ou `ValueError`, avec un message.

f) Proposer une fonction `strCycle( tab : list ) -> str` qui affiche le tableau des villes par leur noms.

e.g. `strCycle( [0,2,1,0] )` renverra la chaîne "Brest-Strasbourg-Marseille-Brest".

g) Créer un tableau de tableaux `PRIX` qui contient le tableau des prix avec les villes prises dans l'ordre alphabétique.

e.g. avec notre exemple cela donnerait `PRIX = [[0, 183.47, 156.77], [183.47, 0, 121.74], [156.77, 121.74, 0]]`. Pour avoir le prix entre Brest (0) et Strasbourg (2) il faut donc appeler `PRIX[0][2]`, tandis qu'entre Strasbourg (2) et Marseille (1) faire `PRIX[2][1]`.

h) Proposer une fonction `arrondiSup( tab2tabPrix : list ) -> None` qui arrondi à l'euro supérieur tous les prix contenus dans le tableau de tableaux `tab2tabPrix`. On pourra utiliser la fonction `math.ceil()`.

i) Proposer une fonction `def verifTab2prix( tab2tab : list ) -> bool` qui renvoie `True` si le tableau de tableaux ne contient que des entiers, sinon lève une exception `TypeError` avec un message.

j) Proposer une fonction `nextVille( ville : int, villesDejaVisitees : list, tabPrix : list ) -> tuple` qui, à partir d'une ville renvoie l'indice de la prochaine ville non encore visitée ayant la jonction la moins élevée, ainsi que le prix de cette jonction.

On pourra utiliser l'infini `float( 'inf' )`.

e.g. `nextVille(0, [0], [0,184,157])` renverra le tuple (2, 157)

e.g. `nextVille(1, [1,2], [184,0 ,121])` renverra le tuple (0, 184)

k) Écrire une fonction `cycleGloutonDepuisVille( villeD : int, tab2tabPrix : list ) -> tuple` qui à partir d'une ville de départ renvoie le cycle de prix minimal selon un algorithme glouton, ainsi que son prix total.

l) Écrire une fonction `main() -> None` qui répond au problème.

m) Faire tourner votre code sur un exemple avec 5 villes et préparer le compte-rendu.

n) Approfondissement :

- ◇ faire une fonction `tests()` qui teste les fonctions et rattrape les exceptions si certains tests en lèvent.
- ◇ modifier votre code afin que le critère pour choisir la prochaine ville ne soit pas la prix mais le nombre minimal de devises pour payer. Le résultat change-t-il ?

Rendu

À la fin du temps imparti, chaque groupe rendra 2 travaux :

a) un compte-rendu (au maximum 2 pages) contenant l'organigramme du code général, un résumé de votre projet, les jeux de tests effectués ainsi que leurs résultats.

b) le fichier contenant le code `python` sera envoyé sous format numérique via l'ENT