

## Epreuve Regroupée

Novembre 2023

**BRANCHE : ALGORITHMES ET LANGAGE DE  
PROGRAMMATION OBJET (ALP)**

**CLASSE : ESIG1 Classes A et B  
(Groupes 1, 2 et 3)**

**DATE : 30 octobre 2023**

**NOM : .....**

**PRÉNOM : .....**

**PROFESSEUR : Eric BATARD / Jasmina TRAVNJAK / Clément VOGT**

**N° du poste de travail : ESIG-PB.....**

**N° de clé USB : .....**

## Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
- Documentation électronique (clé USB, ...) : Autorisée si recopiée avant le début de l'épreuve.
- Tout partage de ressources de votre poste de travail avec le réseau ou toute autre tentative de communication (que ce soit avec un humain ou un outil d'intelligence artificielle) seront considérés comme de la fraude et sanctionnés par la note minimale.

Cela comprend la présence d'un téléphone portable, montre connectée, lunettes connectées, clé USB ou tout autre dispositif de stockage ou de communication à proximité immédiate de votre place de travail : tout ceci doit être posé à l'endroit indiqué par le/la surveillant-e.

## Pour démarrer

- *Avant* le début de l'épreuve, votre documentation sur disque externe, clé USB, ... doit être recopiée sur **C:\ESIGUsers\Doc**. Pour cela, connectez-vous au réseau sur le poste de travail qui vous a été attribué.
- *Une fois l'épreuve commencée*, copiez dans **C:\ESIGUsers** le contenu du dossier réseau qui vous sera indiqué au début de l'épreuve de façon à avoir un répertoire **C:\ESIGUsers\Eléments ER ALP 30-10-23**
- Ouvrez ce dossier **C:\ESIGUsers\Eléments ER ALP 30-10-23**.
- Ouvrez Thonny puis ouvrez les fichiers indiqués dans l'énoncé. *N'oubliez pas d'indiquer vos nom et prénom au début des fichiers modifiés.*

## Consignes générales

### Sur l'organisation

- Lisez tous les documents fournis.
- Complétez les procédures/fonctions/méthodes ou classes demandées de manière à ce qu'elles répondent aux spécifications de l'énoncé.
- Vous rendrez les fichiers correspondants sur la clé USB qui vous sera remise quand vous serez prête à rendre.

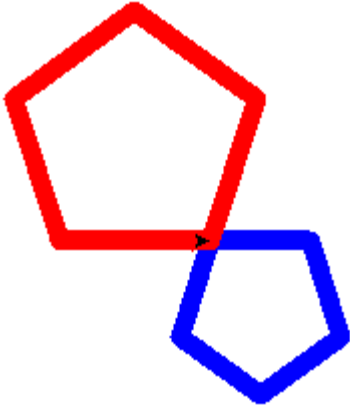
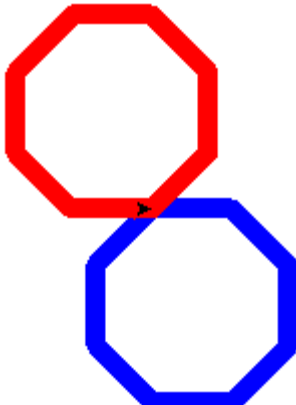
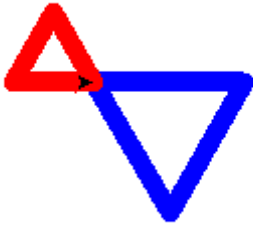
### Sur la programmation en Python

- **Interdiction** d'utiliser les fonctions prédéfinies `max`, `min`, `index`, `count`, `sum`, `mean` ou équivalent dans des modules fournis par python.
- Les éléments suivants sont explicitement autorisés : la fonction `len`, les *slices*, le mot clé `in` sous toutes ses formes, les « compréhensions ». En cas de doute, demandez au responsable d'énoncé.
- Pas de résultats ou de données codés « en dur » : votre code doit pouvoir s'adapter à d'autres jeux de données.
- Les seules variables globales autorisées sont les constantes.

<p>C'est à vous de vérifier que les fichiers enregistrés et rendus contiennent bien la dernière version de votre travail</p>
--

## Partie A – Duo de polygones

L'objectif est de définir une procédure `dessiner_duo_polygones()` permettant d'afficher des dessins comme ceux des exemples ci-dessous. Il faudra aussi procéder à une validation des données. A faire dans le fichier fourni `ER_ALP_Nov23_Tortue.py`

Duo de polygones Nombre de côtés ? 5 Taille d'un côté du premier polygone ? 50 Taille d'un côté du second polygone ? 75	Duo de polygones Nombre de côtés ? 8 Taille d'un côté du premier polygone ? 40 Taille d'un côté du second polygone ? 40	Duo de polygones Nombre de côtés ? 3 Taille d'un côté du premier polygone ? 75 Taille d'un côté du second polygone ? 40
		

Comme son nom l'indique (en grec en tout cas), un polygone possède plusieurs côtés. La première donnée est donc le *nombre de côtés*.

En regardant les exemples, vous pourrez constater que les deux polygones ont toujours le même nombre de côtés mais les côtés n'ont pas toujours la même taille. Les deux autres données seront donc la *taille d'un côté* pour le *premier polygone* et la *taille d'un côté* du *second polygone*.

Enfin il est clair que le polygone « du bas » (le premier dessiné) est toujours tracé en bleu et l'autre en rouge. On souhaite cependant que votre programme soit écrit de telle sorte que la couleur de chacun des deux polygones puisse devenir une donnée saisie par l'utilisateur. Pour cette épreuve, prévoyez juste de passer en paramètre les deux couleurs à la procédure `dessiner_duo_polygones()` mais en indiquant toujours les valeurs 'blue' et 'red' au moment de l'appel de cette procédure. Elle prendra donc en tout 5 paramètres.

En s'inspirant du code de la procédure fournie `polygone_de_base()`, comment obtenir ce dessin ?

La première étape est d'obtenir l'angle duquel la tortue doit tourner. La formule est simple :

$$angle = \frac{360}{nombre\_de\_côtés}$$

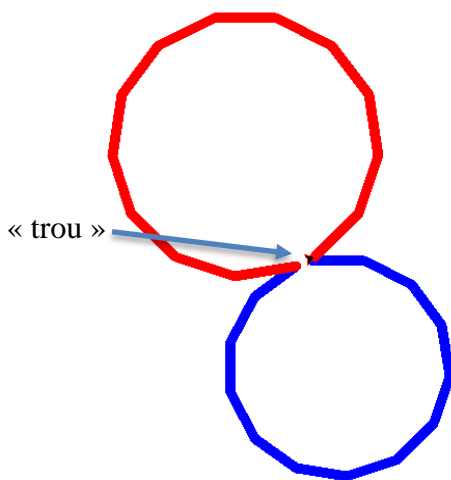
Le premier polygone est alors une application immédiate de l'exemple fourni (procédure `polygone_de_base()`). Le second, quant à lui, est dessiné « à l'envers », en partant en *arrière* contrairement au premier qui est dessiné en partant en *avant*.

Les mots anglais correspondants sont connus : `forward` ou `backward`. Ce sont des procédures déjà définies que vous pouvez utiliser avec la taille du côté en paramètre.

Notez cependant qu'il sera peut-être plus simple pour vous de toujours utiliser `forward()` mais pour aller en arrière il faut donner l'opposé de la taille. C'est-à-dire -50 pour une taille de 50, par exemple. `forward(-50)` est ainsi équivalent à `backward(50)`.

Il est nécessaire de valider les données car trois problèmes peuvent survenir :

- Le nombre de côtés doit à l'évidence être supérieur ou égal à 3 pour un dessin qui fasse sens.
- Si la taille et/ou le nombre de côtés ont une valeur trop grande, le dessin risque de ne pas tenir en entier dans la fenêtre d'affichage de la tortue. On a remarqué que si le produit (= multiplication) de la taille par le nombre de côtés est inférieur ou égal à 1300, le dessin ne dépassera pas les limites de la fenêtre.
- Quand l'angle n'est pas un entier, on risque d'avoir un « trou » à la fin (cf. figure ci-dessous avec 13 côtés).



Pour éviter cela, on va interdire les nombres de côtés qui ne divisent pas 360, c'est-à-dire les nombres tels que le reste de la division de 360 par ce nombre est non nul.

Exemples :

- 6 divise 360 car le reste de la division de 360 par 6 est 0 (cette valeur donnera une figure à 6 côtés, un hexagone, avec un angle de  $60^\circ$  entre chaque paire de côtés)
- 13 ne divise pas 360 car le reste de la division de 360 par 13 est 9.

Partant de ces contraintes, on demande une fonction de validation qui prend en paramètre les trois données saisies par l'utilisateur et qui renvoie `True` ou `False` selon que toutes les données sont valides ou si une au moins ne l'est pas. On veut également les messages d'erreur illustrés ci-dessous en gras (grossis pour plus de lisibilité, à afficher sur une seule ligne).

<i>Erreur a) sur les côtés</i>	<i>Erreur b) sur les côtés</i>	<i>Erreur c) 1<sup>er</sup> polygone</i>	<i>Erreur c) 2nd polygone</i>
Duo de polygones Nombre de côtés ? 2 Taille d'un côté du premier polygone ? 100 Taille d'un côté du second polygone ? 100  <b>Pas assez de côtés</b>  Au moins une donnée erronée : pas de dessin !	Duo de polygones Nombre de côtés ? 13 Taille d'un côté du premier polygone ? 100 Taille d'un côté du second polygone ? 50  <b>Le nombre de côtés ne divise pas le cercle</b>  Au moins une donnée erronée : pas de dessin !	Duo de polygones Nombre de côtés ? 15 Taille d'un côté du premier polygone ? 100 Taille d'un côté du second polygone ? 20  <b>La taille totale du premier polygone dépasse la limite</b>  Au moins une donnée erronée : pas de dessin !	Duo de polygones Nombre de côtés ? 120 Taille d'un côté du premier polygone ? 10 Taille d'un côté du second polygone ? 20  <b>La taille totale du second polygone dépasse la limite</b>  Au moins une donnée erronée : pas de dessin !

On ne fera évidemment aucun dessin si au moins une des données est erronée. Il faudra donc exécuter l'instruction d'initialisation (procédure `init()` fournie), votre procédure `dessiner_duo_polygones()` et l'instruction de finalisation (`done()` prédéfinie) *uniquement* si la fonction de validation a donné son feu vert.

En tout cas, l'instruction `init()` doit apparaître *avant* vos instructions de dessin et `done()` *après*.

## Bienvenue à la **GROZMY** !

Une nouvelle chaîne de supermarchés spécialisée dans le *hard-discount* arrive en Suisse, **Grozmy**.

Cette chaîne est déjà active dans les pays de l'Est (son fondateur est d'origine tchèque) et elle pratique une politique de prix très agressive, basée sur diverses réductions offertes aux détenteurs et detentrices de la célèbre carte **GroBonus**.

### *Principe général des exercices de cette partie*

Vous travaillerez toujours avec au moins une liste. Cette liste est une liste d'achats contenant le montant des divers achats d'un client.

La première question demande simplement de calculer le total de ces achats. Dans la seconde question, vous calculerez également un prix total par catégorie d'articles (cf. les explications détaillées plus loin).

A l'exception de ces deux exercices, les exercices suivants (indépendants les uns des autres) correspondent chacun à une façon d'appliquer une réduction à partir d'une liste d'achats. Cela permettra aux gestionnaires de la Grozmy d'évaluer l'impact de telle ou telle réduction.

L'ensemble des sorties des exercices pour les trois jeux de données fournis est en Annexe.

### *Le fichier fourni* ER\_ALP\_Nov23\_Grozmy.py

Vous complétez ce fichier à partir de ce qui est fourni (qu'il ne faudra surtout pas modifier !) :

- Trois listes d'achats distinctes sont fournies ainsi que, pour la Question 2, les listes de catégorie correspondantes.
- Une fonction `arrondi2()` est fournie pour vous permettre d'afficher les prix arrondis avec toujours deux chiffres après la virgule.
- La procédure `main()` est *entièrement* fournie donc **A NE SURTOUT PAS MODIFIER !**

Vous constaterez que la procédure `main()` appelle trois fois la procédure fournie `test_global()` avec une liste d'achats et une liste de catégories. Cette procédure appelle toutes les procédures d'affichage demandées, donc **A NE SURTOUT PAS MODIFIER !**



## Question 1 – total des achats

Complétez la fonction `total_achats()` qui prend en paramètre une liste d'achats et renvoie la somme de tous les montants.

Complétez également la procédure `afficher_total_achats(liste_achats)` qui prend en paramètre une liste d'achats, appelle la fonction `total_achats()` pour produire finalement l'affichage suivant (exemple avec la première liste d'achats)

```
Le montant total (sans réduction) est de 1569.50 CHF
```

## Question 2 – total par catégories

Chaque achat correspond à une catégorie de produits. Le nom des catégories n'a pas d'importance ici (on verra comment gérer cet aspect à la reprise du cours). Il suffit de savoir que :

- le nombre de catégories est de 4 (une constante `NOMBRE_DE_CATEGORIES` est fournie),
- les catégories sont numérotées de 1 à 4,
- une liste de catégorie donne les catégories des achats figurant dans la liste d'achats correspondante. Ce sont des listes dites *parallèles* où le même indice (position, rang) donne accès à un achat et à sa catégorie selon la liste.

Complétez la fonction `total_par_categorie()` qui prend en paramètre une liste d'achats, la liste de catégories correspondantes et une catégorie (c'est-à-dire un entier entre 1 et 4) et renvoie la somme des montants des achats de cette catégorie uniquement.

Pour tester cette fonction, vous complèterez également la procédure `afficher_toutes_les_categories(liste_achats, liste_categories)` qui prend en paramètre une liste d'achats et la liste de catégories correspondante et appelle la fonction `total_par_categorie()` pour chacune des quatre catégories (une boucle est demandée) afin de produire l'affichage suivant (exemple avec la première liste d'achats et de catégories) :

```
Le montant total pour la catégorie 1 est de 425.70 CHF
Le montant total pour la catégorie 2 est de 283.40 CHF
Le montant total pour la catégorie 3 est de 699.00 CHF
Le montant total pour la catégorie 4 est de 161.40 CHF
```

## Question 3 – calcul des bons GroBonus

Pour les personnes qui détiennent une carte GroBonus, chaque achat à la Grozmy permet de cumuler des points qui sont convertis en bons d'achat. La règle est très simple et, pour simplifier encore, vous n'allez calculer les bons obtenus que sur le total des achats effectués, sans tenir compte d'un reliquat (reste) antérieur.

La règle est : pour chaque tranche de 400 CHF, un bon de 5 CHF est édité.

Notez qu'il faut travailler sur un total entier. Pour cela, appliquez au total des achats la fonction `int()`, la même que celle dont vous avez l'habitude pour les inputs d'entiers.

*Exemple avec la première liste d'achats :*

Le total des achats étant 1569.50 CHF, on travaille sur 1569. On a  $1569 = 3 \times 400 + 369$ .

On peut donc obtenir 3 bons de 5 CHF et 369 points seront ajoutés au compte.

Ce qui aboutit à l'affichage suivant :

Vous recevrez 3 bons de réduction à 5 CHF chacun et vous aurez 369 points de plus sur votre compte.

Vous noterez que, pour cette question, aucune fonction n'est prévue et vous pourrez programmer votre réponse directement dans la procédure `afficher_bons(liste_achats)`.

Dans le cas où on ne peut émettre aucun bon, c'est-à-dire quand le total des achats est plus petit que 400 CHF, par exemple 132 CHF, on affiche le message suivant :

Vous ne recevrez aucun bon de réduction mais vous aurez 132 points de plus sur votre compte.

### Question 4 – Le moins cher offert

On souhaite dans cette question traiter l'action « Le moins cher offert ». Il faut donc trouver le prix de l'article le moins cher acheté dans la liste des achats. C'est l'objet de la fonction à compléter `minimum()` qui prend en paramètre une liste d'achats et qui renvoie l'achat le moins cher.

Vous complèterez aussi la procédure `afficher_minimum(liste_achats)` qui prend en paramètre une liste d'achats, qui appelle la fonction `minimum()` et qui produit l'affichage suivant (exemple pour la première liste d'achats) :

Le montant total avec le moins cher offert est de 1532.50 CHF - vous gagnez : 37.00 CHF

### Question 5 – Réduction au-delà d'un certain montant total

Dans ce cas, on prévoit d'appliquer une réduction si la somme des achats dépasse un certain montant. Notez bien que la réduction, qui est de 10%, ne s'applique qu'aux articles au-delà de cette limite.

Prenons l'exemple de la première liste d'achats avec un montant de 1000 CHF :

<i>position</i>	0	1	2	3	4	5	6	7	8	9
<i>montant</i>	138.50	312.50	312.50	72.00	283.40	152.00	135.20	89.40	37.00	37.00
<i>somme partielle</i>	138.50	451.00	763.50	835.50	<b>1118.90</b>	<i>1270.90</i>	<i>1406.10</i>	<i>1495.50</i>	<i>1532.50</i>	<i>1569.50</i>

La dernière ligne correspond à la somme partielle :

- en position 1, c'est la somme du montant de l'achat 0 et de l'achat 1,
- en position 2, c'est la somme du montant de l'achat 0, de l'achat 1 et de l'achat 2,
- et ainsi de suite.

On voit ici que c'est à partir de l'achat en position 4 qu'on dépasse 1000 CHF. Les achats suivants (en *italique*) bénéficient alors d'une réduction de 10 %

Cela se traduit par l'affichage suivant :

Avec la limite de 1000 CHF, vous ne paierez que 1524.44 CHF - vous gagnez : 45.06 CHF

Pour l'obtenir vous devrez compléter la fonction `reduction_au_dela()` qui prend comme paramètres une liste d'achats et une somme limite. Cette fonction calcule le montant à payer en intégrant la réduction.

C'est la procédure `afficher_reduction_au_dela(liste_achats, limite)` qui va assurer l'affichage et appellera la fonction `reduction_au_dela()`. Le prix réduit sera affiché ainsi que la gain, obtenu par la différence entre le prix total sans réduction et le prix réduit.

Au cas où la limite ne serait pas du tout atteinte, le message devient :

Avec la limite de 1000 CHF, il vous manque 164.50 CHF d'achat pour bénéficier de cette action

## Question 6 – Le second à moitié prix

Dans cette dernière question, on prévoit d'appliquer une réduction si le client achète deux fois le même article. Le premier article coûte le prix normal, le second la moitié du prix normal.

Pour simplifier, nous allons considérer que si deux achats *consécutifs* dans la liste ont le même montant, il s'agit de l'achat de deux fois le même article et donc la réduction s'applique.

Il n'est pas utile de vérifier si trois montants consécutifs ont le même prix car dans ce cas la réduction ne s'applique qu'une fois.

En revanche si on a deux fois de suite deux fois le même montant, la réduction s'appliquera deux fois. Ainsi c'est plus simple à programmer : il suffit de détecter si deux montants consécutifs ont la même valeur. Quand ce sera le cas, il suffira de passer à la suite, c'est-à-dire le montant après le deuxième montant identique (s'il existe dans la liste).

Par exemple avec la première liste d'achats :

<i>position</i>	0	1	2	3	4	5	6	7	8	9
<i>montant</i>	138.50	<b>312.50</b>	<b>312.50</b>	72.00	283.40	152.00	135.20	89.40	<b>37.00</b>	<b>37.00</b>

On remarque que les achats en position 1 et 2 sont identiques, ainsi que les achats en position 8 et 9. On devrait donc gagner  $(312.50 + 37) / 2 = 174.75$ , ce qui sera indiqué par l'affichage :

Avec l'action 2ème à moitié prix, vous ne paierez que 1394.75 CHF - vous gagnez : 174.75 CHF

Pour répondre à cette question, il faudra compléter la définition de la fonction `reduction_par_deux()` qui prend en paramètre une liste d'achats et qui renvoie la somme à payer en tenant compte de la réduction « second à moitié prix ».

Il faudra aussi compléter la procédure `afficher_reduction_par_deux(liste_achats)` qui assurera l'appel de la fonction et procédera à l'affichage selon la présentation ci-dessus.

**Bon travail !**