ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

Épreuve d'Algorithmes et Langage de Programmation
Novembre 2021

Branche : ALGORITHMES ET LANGAGE DE PROGRAMMATION (ALP)

Classes : ESIG 1 Classes A et B (groupes 1, 2 et 3)

Date : 8 novembre 2021

Nom étudiant·e : .....

Prénom étudiant·e : .....

Professeur : .....

Numéro du poste : .....

Numéro de la clé USB : .....

Modalités

1. Durée : 240 minutes en tout.
2. Travail individuel.
3. Documentation personnelle (livres, papiers et électronique) : autorisée.
4. Tout partage de ressources de votre poste de travail avec le réseau ainsi que toute autre tentative de communication seront considérés comme de la fraude et sanctionnés comme tels par la note minimale. Cela comprend la présence de clés USB personnelles, smart watches ou similaires à proximité immédiate de votre place de travail.

Démarrage et préparation

1. ***Avant*** de commencer l'épreuve, votre documentation électronique (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.
2. ***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\ALP-novembre-21**.
3. Inscrivez vos prénom et nom en commentaire au début de chaque fichier **`.py`**

Exercice T1 – Tortue : suite de carrés multicolores

a) Carrés successifs

Le but de cet exercice est de dessiner une suite de carrés à l'aide de la libraire **`turtle`**.

Complétez le fichier **`ERexT1-carres.py`** pour que le programme dessine une suite de carrés concentriques. Le premier carré à une longueur de côté de 10. Le carré suivant a une longueur des côtés 50% plus grande que le précédent. Le dessin s'arrête quand la surface dépasse une limite spécifiée par l'utilisatrice[1].

Pour obtenir la surface d'un carré, on multiplie la longueur d'un côté par elle-même (10*10 = la surface du premier carré).

Votre programme doit afficher un message d'erreur si la limite saisie par l'utilisatrice est strictement plus petite que 100 et ne pas effectuer le dessin.

Utilisez les fonctions/procédures fournies **`aller_a_position_initiale`** et **`input_limite`**. La procédure **`aller_a_position_initiale`** déplace la tortue en haut à gauche : cela laisse plus de place pour le dessin. Elle ne prend pas de paramètre et ne retourne rien. La fonction **`input_limite`** demande à l'utilisatrice de saisir une valeur pour la limite et la retourne. Elle ne prend pas de paramètre.

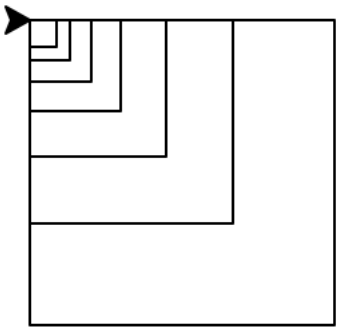
Exemples d'exécution :

Pour une limite de 99, le message d'erreur suivant est produit et il n'y a aucun dessin :

```
Saisir la surface à ne pas dépasser: 99
La limite saisie [ 99 ] est trop petite. Elle doit être au moins: 100
```

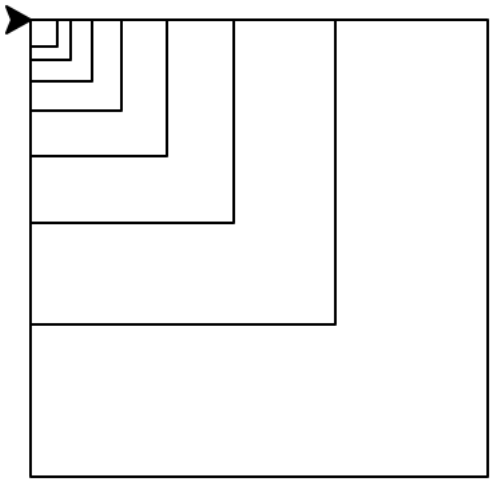
Pour une limite de 100, le dessin suivant est produit (pas de message d'erreur) :

Pour une limite de 29'000 :



[1] Dans le présent document, les termes employés pour désigner des personnes sont pris au sens générique; ils ont à la fois valeur d'un féminin et d'un masculin ou de toute autre forme de genre.

Pour une limite de 29’200



b) Couleurs personnalisables

Complétez le programme afin de permettre à l’utilisatrice de saisir une liste de trois couleurs et les applique aux carrés en alternance.

Une fonction **`input_couleurs`** est fournie. Elle demande à l’utilisatrice de saisir cette liste en séparant les noms de couleur par un espace puis la retourne. Utilisez-la ! Vous n’avez pas besoin de comprendre comment elle effectue son travail, il suffit de l’appeler. Par exemple, **`couleurs = input_couleurs()`** stockera dans la variable **`couleurs`** la liste qui nous intéresse.

Le programme doit vérifier que la liste contient exactement trois éléments avant de faire un dessin, en plus de la validation de la surface déjà réalisée. Un message doit être affiché pour chaque erreur rencontrée.

Exemples d’exécution (voir PDF pour une version en couleur) :

Limite invalide, couleurs valides

```
Saisir la surface à ne pas dépasser: 0
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green red blue
La limite saisie [ 0 ] est trop petite. Elle doit être au moins: 100
```

Limite valide, couleurs invalides

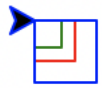
```
Saisir la surface à ne pas dépasser: 100
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green red
Il faut saisir exactement trois couleurs.
```

Limite et couleurs invalides

```
Saisir la surface à ne pas dépasser: 0
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green red
La limite saisie [ 0 ] est trop petite. Elle doit être au moins: 100
Il faut saisir exactement trois couleurs.
```

Limite 1000, couleurs : green red blue

```
Saisir la surface à ne pas dépasser: 1000
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green red blue
```

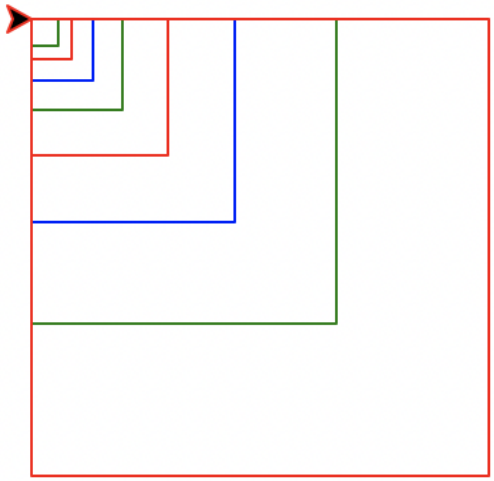


Limite 1000, couleurs : green yellow green

```
Saisir la surface à ne pas dépasser: 1000
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green yellow green
```

Limite 29200, couleurs : green red blue

```
Saisir la surface à ne pas dépasser: 29200
Saisir une liste de 3 couleurs séparées par un espace (p.ex. green red
blue): green red blue
```



Exercice 1 – *Battle royale* – duels successifs

Nous allons simuler un jeu simplifié du type « Battle royale », c'est dire une compétition tous contre tous avec une seule gagnante.

Les règles de base de ce jeu sont comme suit : un certain nombre de joueuses (**`nb_joueuses`**) vont s'affronter. À chaque tour, des duels sont organisés et les vainqueurs passent au tour suivant. En cas de nombre de joueuses impair, l'une est éliminée par tirage au sort. Le jeu s'arrête s'il ne reste plus qu'une seule joueuse.

Par exemple, si **`nb_joueuses = 10`** il y a aura 5 duels au premier tour et donc encore 5 joueuses au deuxième tour. Comme 5 est un nombre impair, une personne est éliminée et deux duels sont organisés. N'en resteront que 2 joueuses au troisième tour donnant lieu à un dual final qui déterminera la gagnante. Le nombre de joueuses évolue donc comme suit : 10 → 5 → 2 → 1.

Ce sont les exceptions aux règles de base qui souvent rendent les jeux intéressants. Ici, il en existe deux :

- D'une part, le *nombre initial de joueuses doit être pair* pour éviter d'éliminer quelqu'un de la compétition par tirage au sort avant même d'avoir joué un seul tour. Par ailleurs, il doit être au moins de 8.
- D'autre part, pour éviter d'éliminer quelqu'un par tirage au sort juste avant la fin, *le jeu s'arrête directement s'il ne reste plus que 3 joueurs* et la victoire est alors partagée. Ce cas s'appelle « une triade de la gloire » et, étonnamment, c'est particulièrement prestigieux.

Votre travail consiste à compléter le fichier **`ERex01-duels.py`** pour que le programme affiche l'évolution du nombre de joueuses par rapport à un nombre de joueuses initial **`nb_joueuses`** saisi par l'utilisatrice.

Le programme doit également vérifier que la valeur de **`nb_joueuses`** soit conforme aux règles énoncées et produire un message d'erreur générique sinon.

Conseil : commencez par traiter les arrêts de jeu quand il y a une seule gagnante et gardez le cas de la triade pour la fin.

Voici des exemples de sorties attendues :

Valeur de **nb_joueuses** *saisi*	*Sortie produite*
6	**invalide**
9	**invalide**
8	**Evolution** **8** **4** **2** **1**
10	**Evolution** **10** **5** **2** **1**
32	**Evolution** **32** **16** **8** **4** **2** **1**
30	**Evolution** **30** **15** **7** **3** **Triade !**

Exercice 2 – *Battle royale* – vote de continuation avec niveaux de motivation

Nous allons maintenant regarder une version un peu plus complexe de notre jeu.

Dans une série qui a fait fureur récemment, par exemple, les participantes ont l'occasion de voter l'arrêt du jeu à tout moment.

Nous allons modéliser cela à l'aide d'une liste de *niveaux de motivation* de chaque participante. Ce niveau est un nombre entier entre 1 et 100.

Un niveau strictement supérieur à 50 indique une volonté de continuer et un niveau entre 1 et 50 une volonté d'arrêter. La valeur 0 n'existe pas.

Par ailleurs, cette même liste sert également à marquer les participantes disqualifiées : elles auront une valeur de motivation égale à -1.

Supposons par exemple qu'il n'y ait que 4 participantes et que leurs niveaux de motivation soient exprimés comme suit : **`[51, -1, 50, 1]`**

Si on votait, il y aurait 1 voix pour continuer (51) et 2 pour arrêter le jeu (50 et 1. La deuxième participante étant disqualifiée (-1), son vote ne compte pas.

Votre travail consiste à coder la fonction booléenne **`jeu_continue(motivations)`** du fichier **`ERex02-vote.py`**. Elle détermine si un vote penche en faveur d'une continuation du jeu (valeur retournée **`True`**) ou d'un arrêt (valeur retournée **`False`**) sur la base d'une liste de niveaux de motivation **`motivations`**. Pour que le jeu continue, le nombre de voix pour continuer doit être supérieur ou égal au nombre de voix pour arrêter. Évidemment, si la liste contient uniquement des participantes disqualifiées le jeu s'arrête.

Pour des raisons de débogage, cette fonction doit également afficher les statistiques du vote. Dans l'exemple donné ci-dessus (liste **`[51, -1, 50, 1]`**), le message serait :

```
Le jeu se termine : 1 oui sur 3 votes
```

Pour tester votre programme, différentes listes de motivation ont déjà été préparées et l'utilisatrice n'a qu'à saisir un numéro identifiant le cas.

Vous n'avez besoin ni de valider le numéro saisi (c'est déjà fait), ni de valider le contenu des listes données (elles sont correctes).

Voici les sorties attendues :

Cas de test n°1, motivations = [-1, -1, -1]

```
Saisir numéro de cas de test: 1
Le jeu se termine : 0 oui sur 0 votes
Jeu continue? => False
```

Cas de test n°2, motivations = [98, 10, 99, 83, 50, 37, 70, 8, 10, 19,
61, 100, 32, 46, 50, 10, 24, 98, 71, 59,
28, 32, 81, 67, 2, 44, 35, 67, 50]

```
Saisir numéro de cas de test: 2
Le jeu se termine : 12 oui sur 29 votes
Jeu continue? => False
```

Cas de test n°3, motivations = [98, 10, 99, 83, 50, -1, 70, 8, 10, 19,
61, 100, 32, 46, 50, 10, -1, 98, 71, 59,
28, -1, 81, 67, -1, 44, 35, 67, 51, 50]

```
Saisir numéro de cas de test: 3
Le jeu continue : 13 oui sur 26 votes
Jeu continue? => True
```

Exercice 3 – *Battle royale* – évolution des niveaux de motivation en fonction des gains réalisés

Nous allons ajouter une liste de gains que les joueuses ont déjà réalisés durant le jeu en plus des niveaux de motivation. Ces deux listes sont liées par les indices, c'est dire que **`motivations[i]`** et **`gains[i]`** contiennent des informations relatives à la même participante.

Sur la base de ces informations, nous pouvons calculer l'évolution de la motivation de chaque participante à un moment donné du jeu : plus elle réalise de gains par rapport aux autres, plus la motivation à continuer augmente.

Pour faciliter votre travail, nous avons décomposé cette tâche en deux étapes. Si vous ne réussissez pas l'étape a) vous pouvez quand même faire l'étape b).

a) Calcul du gain minimal réalisé par les joueuses motivées

Il vous faut d'abord calculer le gain *minimal* réalisé parmi les participantes ayant encore la motivation à continuer.

Regardons un exemple correspondant aux données suivantes :

```
motivations = [92, 66, 50, -1,  6]
gains       = [70, 40, 60, 55, 30]
```

Par souci de lisibilité, nous représentons ces listes sous forme de un tableau ci-desssous comprenant également le numéro de la participante concernée pour chaque paire de valeurs :

Numéro participante	1	2	3	4	5
Motivation	92	66	50	-1	10
Gain	70	40	60	55	30

Ici, le gain minimum réalisé parmi les joueuses motivées (numéros 1 et 2) est de 40. En effet, nous ignorons les gains d'une valeur de 60, 55 et 30 parce qu'ils appartiennent à des joueuses qui ne sont plus motivées (motivation <= 50) ou déjà disqualifiées (motivation = -1).

Concrètement, vous allez définir une fonction **`gain_min_motivees`** dans le fichier **`ERex03-evolution-motivation.py`** qui prend en paramètre une liste de niveaux de motivations et une liste de gains réalisés et doit retourner le gain minimal réalisé parmi les joueuses motivées (motivation > 50). Vous allez appeler cette fonction depuis **`main`** et afficher son résultat.

Par ailleurs, pour vous simplifier la tâche, il y aura toujours au moins une joueuse motivée en première position des listes passées en paramètre.

Voici quelques exemples de sorties attendues (la liste des motivations ne concerne que l'étape b):

No. cas	Données	Sorties produites
1	motivations = [51] gains = [10]	**Saisir numéro de cas de test: 1** **gain_min_motivees 10** **motivations [51]**
2	motivations = [51, 99] gains = [10, 20]	**Saisir numéro de cas de test: 2** **gain_min_motivees 10** **motivations [51, 100]**
3	motivations = [80, 30, 51] gains = [30, 10, 20]	**Saisir numéro de cas de test: 3** **gain_min_motivees 20** **motivations [90, 20, 51]**
4	motivations = [92, 66, 50, -1, 6] gains = [70, 40, 60, 55, 30]	**Saisir numéro de cas de test: 4** **gain_min_motivees 40** **motivations [100, 66, 60, -1, -1]**
5	motivations = [66, -1, 60, 11, 95] gains = [41, 70, 41, 30, 55]	**Saisir numéro de cas de test: 5** **gain_min_motivees 41** **motivations [66, -1, 60, 1, 100]**

b) Mise à jour des niveaux de motivation

Vous allez maintenant compléter la définition de la procédure **mettre_a_jour_motivations** dans le même script que pour l'étape précédente. Cette procédure prend en paramètre le gain minimal des joueuses motivées calculé précédemment **gain_min**, ainsi que la liste des motivations **motivations** et la liste des gains **gains**. Elle doit mettre à jour la liste des motivations selon les règles ci-dessous.

Nous allons les illustrer à l'aide du même exemple que pour l'étape précédente :

No. participante	1	2	3	4	5
Motivation	92	66	50	-1	10
Gain	70	40	60	55	30

Le gain minimal (**gain_min**) réalisé par une joueuse motivée dans cet exemple est de 40.

Règles :

- Si le gain d'une participante est strictement supérieur à **gain_min**, sa motivation augmente de 10. Toutefois, la motivation ne peut dépasser 100 et si une participante est disqualifiée sa motivation reste à -1. Dans l'exemple (tableau ci-dessus), la motivation du numéro 1 deviendra 100, celle du numéro 3 deviendra 60 et la motivation du numéro 4 ne bouge pas.
- Si le gain est égal à **gain_min**, la motivation ne change pas. Dans l'exemple, la motivation du numéro 2 reste à 66.
- Si le gain est inférieur à **gain_min**, la motivation diminue de 10. Toutefois, si elle atteint un niveau de 0 ou inférieur, la participante est disqualifiée et sa motivation devient -1. Dans l'exemple, la motivation du numéro 5 devient -1 (10-10 = 0, donc → -1).

Une fois que les niveaux de motivation ont été mis à jour, la procédure doit afficher la liste modifiée en la passant simplement à la fonction **print**. Vous devez appeler la procédure **mettre_a_jour_motivations** depuis **main** en lui passant les paramètres adéquats.

*Conseil : si vous n'avez pas (complètement) réussi l'étape a, vous pouvez écrire différentes valeurs pour **gain_min** directement dans le **main** pour tester votre code.*