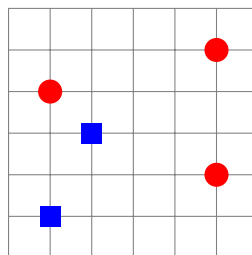


L'essor des big-data et de l'intelligence artificielle est dû à l'apparition de nouveaux algorithmes adaptés à la résolution de problèmes complexes : reconnaissance d'images, comportement des électeurs, conduite autonome des voitures... Dans cette seconde partie tu vas programmer quelques algorithmes emblématiques et innovants.

Cours 1 (Les k voisins les plus proches).

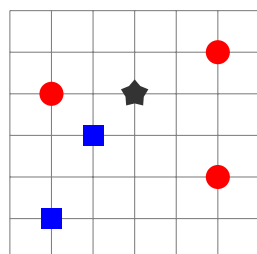
Considérons un électeur qui ne sait pas pour qui voter. Il décide donc de voter comme ses voisins !



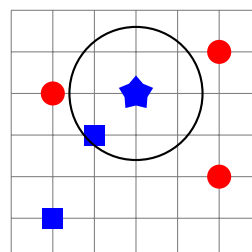
Points de références (rond rouges/carrés bleus)

Pour modéliser la situation, on considère d'abord des points du plan qui peuvent être de deux types, soit des carrés bleus, soit des ronds rouges, on appelle ces points les *points de référence*. Ces points représentent les personnes qui savent déjà pour qui elles vont voter (parmi le choix rouge/bleu). Les autres points n'ont pas encore de couleur, cela représente toutes les personnes qui ne savent pas pour qui elles vont voter.

Voici comment les indécis se décident : pour chaque point non colorié, on regarde le point de référence le plus proche, on colorie alors le point par la couleur de ce point de référence.



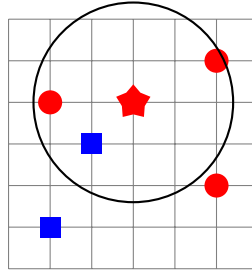
Un indécis (étoile noire)



Coloration de l'indécis par le voisin le plus proche

On généralise cette procédure avec la notion de « k voisins ». Soit k un entier positif. Pour un point non colorié, on cherche les k points de référence les plus proches (ce sont les k voisins). La couleur attribuée est la couleur majoritaire de ces k voisins.

Sur l'exemple ci-dessous, les $k = 3$ voisins sont formés de 2 ronds rouges et 1 carré bleu. On colorie donc l'étoile en rouge.



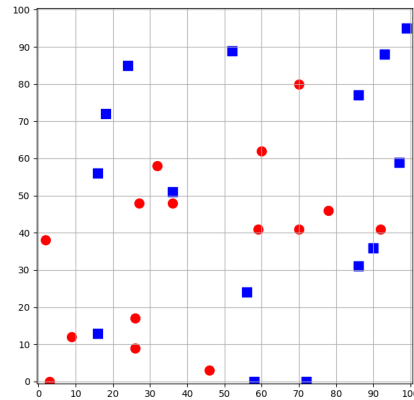
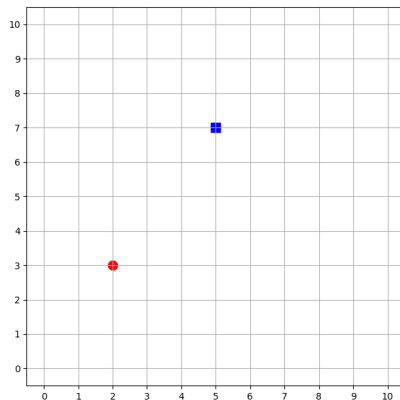
Coloration de l'indécis par
les 3 voisins les plus proches

Le but de l'activité suivante est de colorier tous les points d'une zone grâce à cette méthode.

Activité 1 (Les k voisins les plus proches).

Objectifs : colorier un point en fonction de la couleur de ses voisins.

1. **Préparation.** Programme des fonctions qui permettent d'afficher des points en couleur. On définit un point coloré, appelé *cpoint*, par (x, y, c) où x et y sont des entiers avec $x_{\min} \leq x \leq x_{\max}$ et $y_{\min} \leq y \leq y_{\max}$ et c représente la couleur : $c = 0$ pour du rouge et $c = 1$ pour du bleu.
 - (a) Définis des constantes globales x_{\min} , x_{\max} , y_{\min} , y_{\max} (par exemple 0, 10, 0, 10) pour la fenêtre des points. Programme une fonction `afficher_cpoints(cpoints)` qui affiche une liste de *cpoints*. Par exemple pour `cpoints = [(2,3,0), (5,7,1)]` cette fonction affiche un point en (2,3) en rouge et un point en (5,7) en bleu (figure de gauche).



- (b) Programme une fonction `fonction_couleur(x,y)` qui renvoie une couleur (0 ou 1) pour un point (x, y) . Tu pourras essayer plusieurs fonctions :
 - 0 ou 1 au hasard,
 - 0 si $((x^2 + y^2) \% 100) - 50 > 0$ et 1 sinon,
 - 0 ou 1 selon le signe de $(x - \frac{x_{\max}}{2})^3 - 3(x - \frac{x_{\max}}{2})(y - \frac{y_{\max}}{2})^2 - x_{\max}$,
 - ou toute autre fonction de ton invention...
- (c) Programme une fonction `generer_cpoints(N)` qui renvoie une liste aléatoire de N *cpoints* (x, y, c) (utilise la fonction précédente pour calculer c). Sur la figure de droite ci-dessus on a affiché 30 points sur $[0, 100] \times [0, 100]$.

2. Le voisin le plus proche.

- (a) Programme une fonction `distance(P,Q)` qui calcule la distance entre $P = (x_1, y_1)$ et $Q = (x_2, y_2)$:

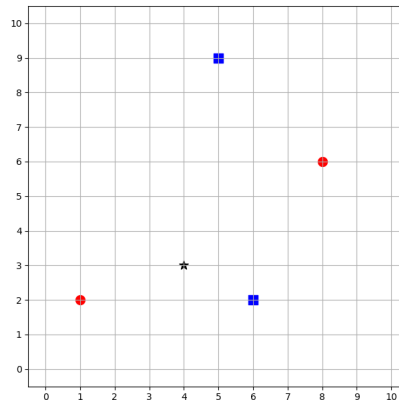
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

- (b) Programme une fonction `un_voisin_proche(P, cpoints)` qui à partir d'un point $P = (x_0, y_0)$ renvoie un cpoint $Q_c = (x, y, c)$ parmi ceux de la liste donnée et qui est le plus proche possible de P .

Indications.

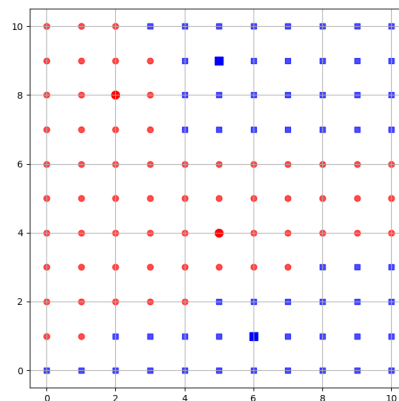
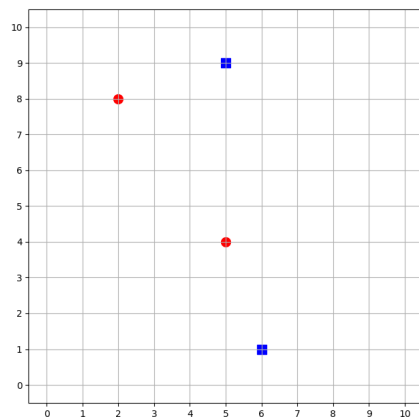
- Il peut y avoir plusieurs voisins à la même distance, la fonction en renvoie un, peu importe lequel.
- Commence par définir un réel d_{\min} très grand (par exemple $d_{\min} = 1000$ ou mieux $d_{\min} = +\infty$ par `dmin = inf` du module `math`). Ensuite calcule la distance entre P et chaque Q_c de la liste et renvoie le plus proche.

Sur l'exemple suivant $P = (4, 3)$ (en noir) a pour plus proche voisin $Q = (6, 2)$ (en bleu) qui est à une distance $d = \sqrt{5}$.

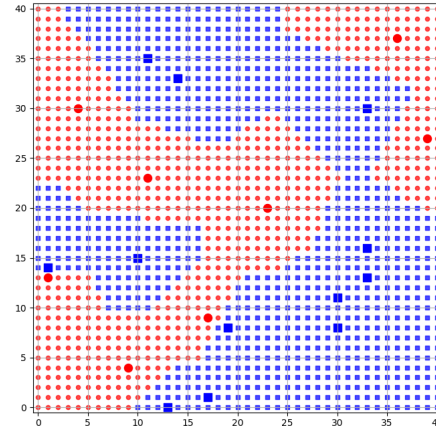
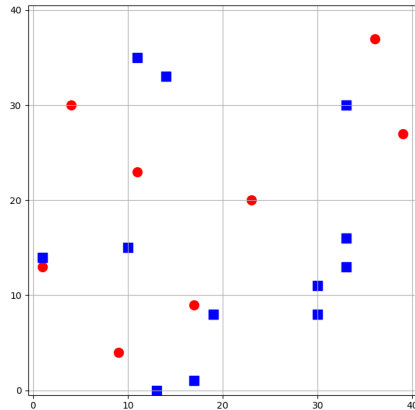


3. **Coloriage.** À partir d'une liste de points colorés, on colorie tous les points : chaque point prend la couleur du voisin le plus proche. Programme ceci en une fonction `colorier_par_un_voisin_proche(cpoints)`.

Ci-dessous à gauche des points colorés initiaux et à droite le coloriage obtenu par le plus proche voisin.

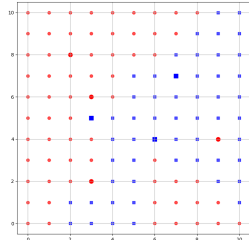
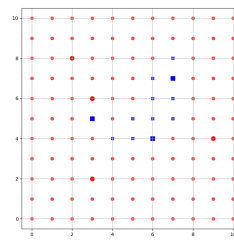
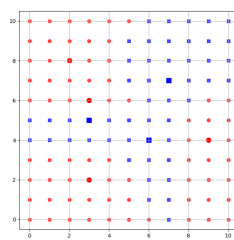
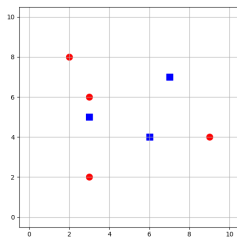


Voici un autre exemple.

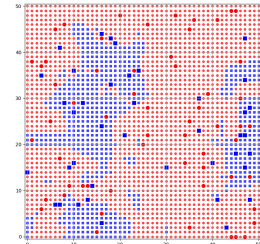
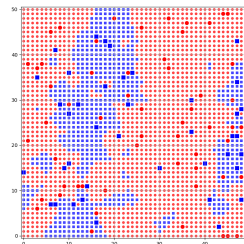
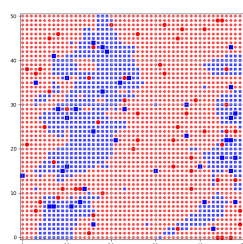
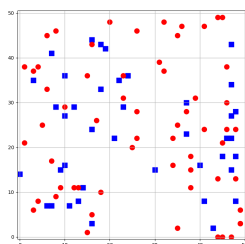


4. Les k voisins proches.

- (a) Programme une fonction `les_voisins_proches(P, cpoints, k)` qui renvoie les k voisins les plus proches du point P .
- $P = (x_0, y_0)$ est un point,
 - `cpoints` est la liste des cpoints (x, y, c) initiaux,
 - $k \geq 1$ est un entier,
 - la fonction renvoie une liste de k cpoints.
- (b) Programme une fonction `couleur_majoritaire(cpoints)` qui renvoie la couleur majoritaire (0 ou 1) d'une liste de cpoints.
- (c) Programme une fonction `colorier_par_les_voisins_proches(cpoints, k)` qui à partir d'une liste de points colorés initiaux, colorie tous les autres points : chaque point prend la couleur majoritaire des k voisins les plus proches ($k = 1$ correspond au voisin le plus proche).
- Ci-dessous à gauche les points colorés initiaux et à droite des coloriages pour différentes valeurs de k ($k = 1, 2, 3$).



Et voici un exemple de taille plus grande avec $k = 3, 5, 7$.



Cours 2 (Expression correctement parenthésée).

Voici des exemples d'expressions bien et mal parenthésées :

- $2 + (3 + b) \times (5 + (a - 4))$ est correctement parenthésée ;
- $(a + 8) \times 3) + 4$ est mal parenthésée : il y a une parenthèse fermante «) » seule ;
- $(b + 8/5)) + (4$ est mal parenthésée : il y a autant de parenthèses ouvrantes « (» que de parenthèses fermantes «) » mais elles sont mal positionnées.

Dans le chapitre « Calculatrice polonaise – Piles » du premier tome, nous avons étudié un algorithme qui vérifie si une expression a ses parenthèses correctes. Cette méthode résout complètement le problème mais nécessite beaucoup de mémoire. En effet la taille de la pile peut être de l'ordre de grandeur de la longueur de l'expression, ce qui pose des problèmes pour des expressions ayant des millions de caractères. Par contre avec la méthode de l'activité suivante tu ne stockes que les valeurs de deux entiers (h et S), l'inconvénient c'est que la réponse renvoyée est probablement vraie, mais ce n'est pas une certitude. Cette méthode permet aussi de vérifier si un fichier (par exemple un fichier « xml ») a un balisage correct.

Activité 2 (Parenthèses correctes?).

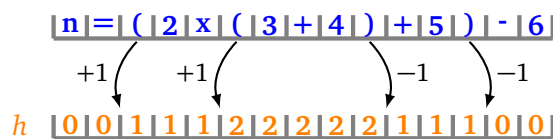
Objectifs : tester si une expression a ses parenthèses et ses crochets bien placés.

Partie A. Parenthèses seules.

On considère une expression avec des parenthèses (tous les autres caractères sont ignorés). On associe une hauteur h en lisant l'expression de gauche à droite :

- au départ $h = 0$,
- avant une parenthèse ouvrante "(" on augmente h de 1,
- après une parenthèse fermante ")" on diminue h de 1.

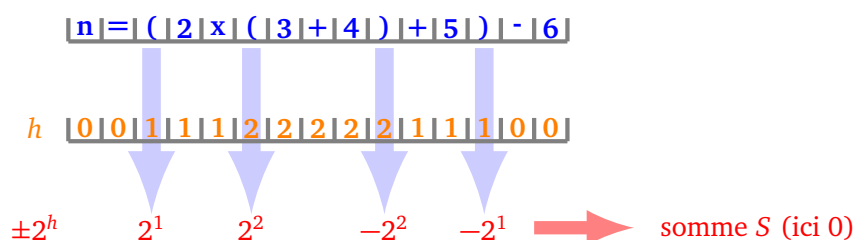
Voici un exemple avec l'expression « $n = (2 \times (3 + 4) + 5) - 6$ ».



On fixe un nombre premier p (par exemple $p = 11$ ou $p = 101$) et on pose $a = 2$. À chaque parenthèse on associe un entier modulo p :

$$" (" \mapsto 2^h \pmod{p} \quad \text{et} \quad ")" \mapsto -2^h \pmod{p},$$

à tous les autres caractères on associe 0. Ensuite on calcule la somme S de tous ces termes.



- Si l'expression est bien parenthésée alors $S = 0$.
- Si à un moment $h < 0$ ou si à la fin $S \neq 0$ alors l'expression est mal parenthésée.

Programme cette méthode en une fonction `test_parentheses(expression)` qui renvoie « Vrai » si la somme de contrôle vaut 0 et « Faux » sinon.

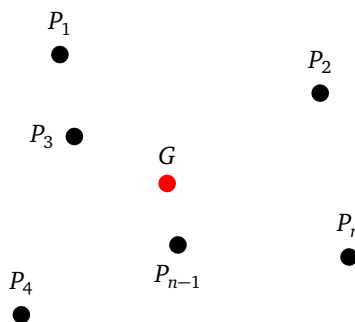
Voici un exemple d'une expression mal parenthésée : « $x(x)x(x(x)x$ » (ici x désigne n'importe quelle suite de caractères autre que des parenthèses), le problème est bien détecté car la somme $S = 2$ n'est pas nulle.

On a une certitude uniquement dans le cas où $S \neq 0$ (l'expression est alors mal parenthésée/crochetée), dans le cas $S = 0$ on a seulement une forte probabilité que l'expression soit bien parenthésée/crochetée. Voici des exemples de situations non détectées. Par exemple "[] ()" (mais si on regarde juste les parenthèses on voit que l'expression n'est déjà pas correcte). Il faut donc au préalable vérifier que les parenthèses seules sont bien positionnées, et aussi que les crochets seuls sont bien positionnés. Autre soucis possible avec "[(" et $p = 11$ alors on trouve une somme $S = 2^1 + 3^2 = 11$ donc $S = 0 \pmod{11}$ et pourtant l'expression est mal parenthésée/crochetée. Par contre avec $p = 7$, on trouve $S = 4 \pmod{7}$, donc cela prouve que l'expression est mal parenthésée/crochetée.

Note. Cet algorithme peut être programmé en temps réel, c'est-à-dire que l'on peut commencer les calculs dès la lecture du premier caractère et les terminer juste après le dernier.

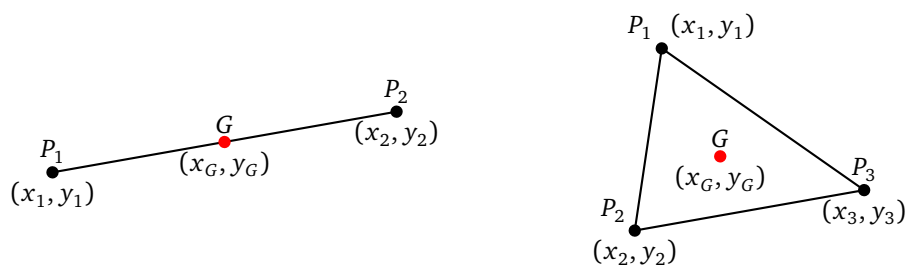
Cours 3 (Isobarycentre).

On considère n points du plan P_1, P_2, \dots, P_n . On cherche un point « le plus au milieu possible » de tous ces points. On appelle ce point **l'isobarycentre** des $\{P_i\}$ et on le note G .



Cas de deux points. Si $n = 2$ alors G est simplement le milieu du segment $[P_1, P_2]$.

Cas de trois points. Si $n = 3$ alors G est le centre de gravité du triangle défini par les trois points. C'est donc l'intersection des médianes.



Formule. Pour n quelconque, voici la formule pour calculer les coordonnées (x_G, y_G) de l'isobarycentre G en fonction des coordonnées (x_i, y_i) des P_i ($i = 1, \dots, n$) :

$$x_G = \frac{x_1 + x_2 + \dots + x_n}{n} \quad \text{et} \quad y_G = \frac{y_1 + y_2 + \dots + y_n}{n}$$

Cela signifie que l'abscisse de G est simplement la moyenne des abscisses des P_i et l'ordonnée de G est la moyenne des ordonnées des P_i . Pour $n = 2$ on retrouve les coordonnées du milieu $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$. Et pour $n = 3$ cela donne

$$(x_G, y_G) = \left(\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3} \right).$$



Activité 3 (Barycentres).

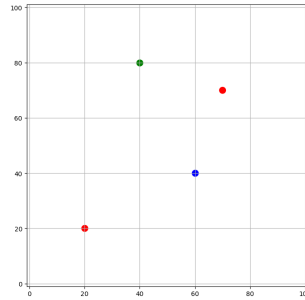
Objectifs : rassembler des points en groupes.

Note. Certaines fonctions sont très proches de la première activité sur les k voisins.

1. Afficher des points.

Programme d'abord une fonction `afficher_points(points, couleurs)` qui affiche une liste de points (x, y) en fonction d'une liste de couleurs (une couleur par point).

Par exemple avec `points = [(20,20), (60,40), (40,80), (70,70)]` et `couleurs = [0,1,2,0]` alors `afficher_points(points, couleurs)` affiche le graphique ci-dessous.

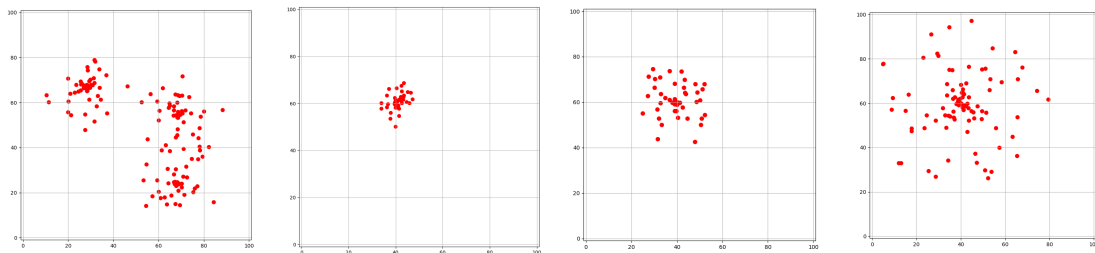


Indications.

- Tu peux définir des constantes x_{\min} , x_{\max} , y_{\min} , y_{\max} pour définir une fenêtre de visualisation. On prendra par défaut $[0, 100] \times [0, 100]$.
- Pour les couleurs, 0 peut être pour le rouge, 1 pour le bleu...

2. Générer des points.

On veut générer des groupes aléatoires de points, comme ci-dessous à gauche avec trois groupes de points. Chaque groupe est généré au hasard autour d'un centre, avec un paramètre de dispersion qui fait que les points s'éloignent plus ou moins du centre (voir les trois figures de droite : un seul groupe de points mais avec des paramètres de dispersion différents).



Voici comment programmer une fonction `generer_points(k, n, d)` qui affiche k groupes, de n points chaque, selon une dispersion d .

Répéter k fois :

- Choisir un centre (x_c, y_c) au hasard dans la fenêtre (c'est mieux s'il n'est pas trop près des bords).
- Pour ce centre, répéter n fois :
 - Choisir un angle θ au hasard entre 0 et 2π .
 - Choisir un rayon r au hasard selon la formule $r = d\rho^2$ où ρ est un réel tiré au hasard entre 0 et 1.
 - Ajoute le point (x, y) à la liste des points, où :

$$x = x_c + r \cos \theta \quad \text{et} \quad y = y_c + r \sin \theta$$

(vérifie quand même que (x, y) est bien dans la fenêtre voulue).

Indications. La fonction `random()` (sans argument) du module `random` renvoie à chaque appel un nombre flottant aléatoire x de $[0, 1[$. Pour obtenir à partir de x un nombre aléatoire y entre a et b , on peut utiliser la formule :

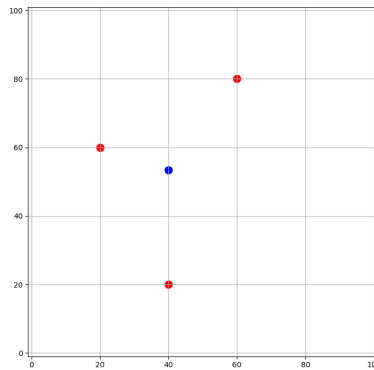
$$y = a + (b - a)x.$$

3. Calcul du barycentre.

Programme une fonction `calcul_barycentre(points)` qui calcule l'isobarycentre (x_G, y_G) d'une liste de points $\{(x_i, y_i)\}$. On rappelle la formule :

$$x_G = \frac{x_1 + x_2 + \dots + x_n}{n} \quad \text{et} \quad y_G = \frac{y_1 + y_2 + \dots + y_n}{n}.$$

Exemple. Avec les points $(20, 60)$, $(40, 20)$, $(60, 80)$ (en rouge) la fonction calcule les coordonnées du barycentre $(x_G, y_G) = (40, 53.33\dots)$ (en bleu).



4. Barycentre le plus proche.

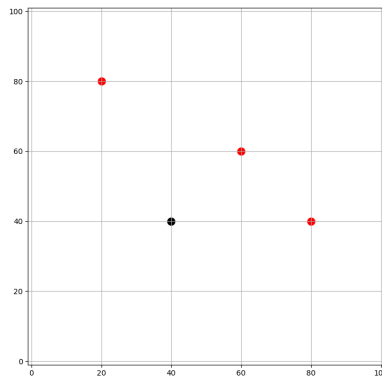
Dans cette question on se donne une liste de barycentres (on verra plus tard comment on les obtient) et une liste de points. Pour chaque point on va trouver quel est le barycentre le plus proche.

- (a) Programme une fonction `distance(P, Q)` qui calcule la distance entre $P = (x_1, y_1)$ et $Q = (x_2, y_2)$:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

- (b) Programme une fonction `barycentre_proche(P, barycentres)` qui à partir d'un point $P = (x_0, y_0)$ renvoie le rang du point le plus proche dans la liste `barycentres` donnée.

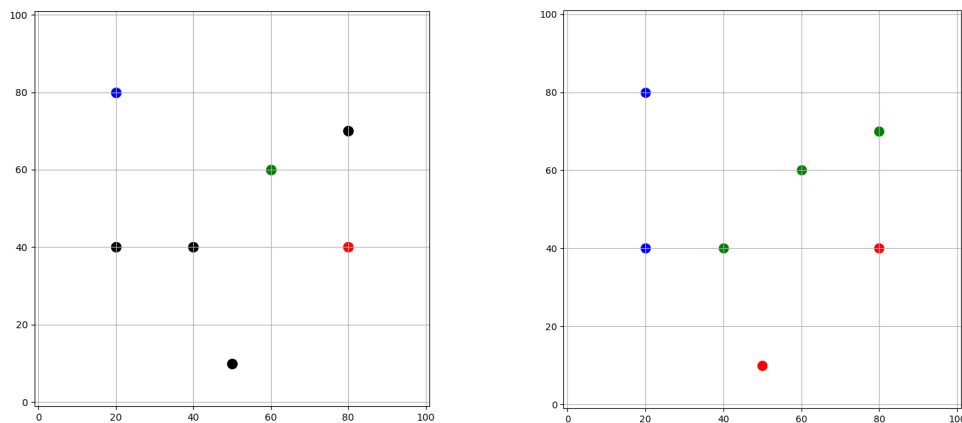
Par exemple si nous avons la liste `barycentres = [(80, 40), (20, 80), (60, 60)]` (en rouge) et le point $P = (40, 40)$ (en noir), alors le barycentre le plus proche de P est $G_2 = (60, 60)$ qui est de rang 2 dans la liste des barycentres. Donc la fonction `barycentre_proche(P, barycentres)` renvoie ici 2.



(c) **Coloriage suivant le barycentre le plus proche.** La fonction précédente renvoie le rang du barycentre le plus proche et on considère ce rang comme la couleur (le barycentre G_0 en tête de liste est de couleur 0 donc rouge, le barycentre G_1 est de couleur 1 donc bleu...).

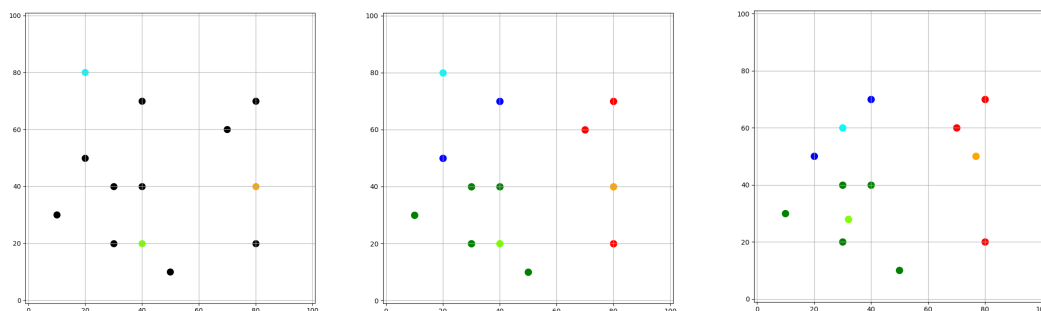
Déduis-en une fonction `couleurs_barycentres_proches(points, barycentres)` qui à partir d'une liste de points renvoie la liste des couleurs attribuées à chaque point (selon le rang du barycentre le plus proche, c'est-à-dire la couleur).

Exemple. Reprenons nos barycentres `barycentres = [(80,40), (20,80), (60,60)]` avec chacun une couleur donnée par son rang : $G_0 = (80,40)$ est de couleur 0 (rouge), $G_1 = (20,80)$ est de couleur 1 (bleu), $G_2 = (60,60)$ est de couleur 2 (vert). Fixons aussi une liste de points `points = [(40,40), (20,40), (80,70), (50,10)]` pour l'instant sans couleur (en noir sur la figure de gauche ci-dessous). Alors la fonction `couleurs_barycentres_proches()` renvoie la liste `[2, 1, 2, 0]`. Cela signifie que le point $P_0 = (40,40)$ doit être colorié par la couleur 2 car le plus proche barycentre est G_2 , le point $P_1 = (20,40)$ doit être colorié par la couleur 1 car le plus proche barycentre est G_1 ... Cela permet de colorier les quatre points (figure de droite).



5. Recalculer les barycentres.

- On se donne une liste de points et une liste de barycentres (figure de gauche ci-dessous, pour l'instant cette liste de « barycentres » est arbitraire).
- On a vu comment attribuer une couleur à chaque point (figure centrale), ainsi les points sont regroupés par couleur. Les points bleus sont associés au barycentre bleu clair, les points verts sont associés au barycentre vert clair...
- Pour tous les points d'une même couleur, on calcule le barycentre. On se retrouve avec une nouvelle liste de barycentres (figure de droite) (cette fois ce sont de « vrais » barycentres).



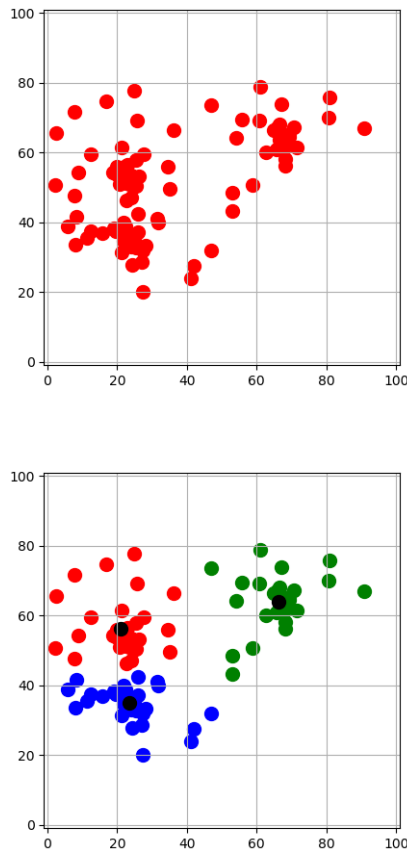
Programme une fonction `recalculer_barycentres(points, barycentres)` qui effectue cette tâche. Sur l'exemple graphique ci-dessus les barycentres de départ sont `[(80,40), (20,80), (40,20)]`

(figure de gauche), la fonction `recalculer_barycentres()` renvoie les coordonnées des nouveaux barycentres : $[(76.66\dots, 50), (30, 60), (32, 28)]$.

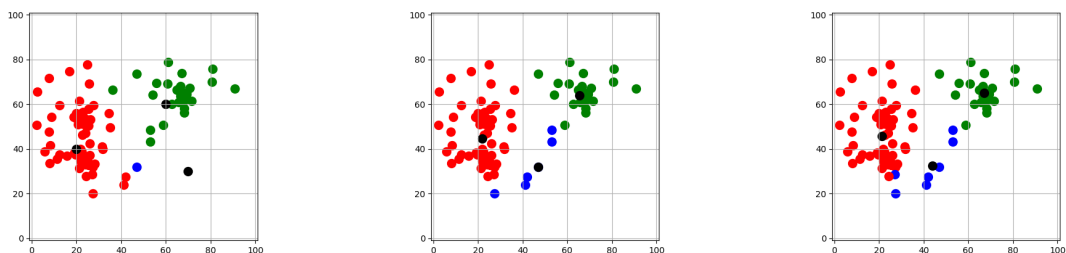
Indications. Si un barycentre n'est associé à aucun point on le conserve inchangé.

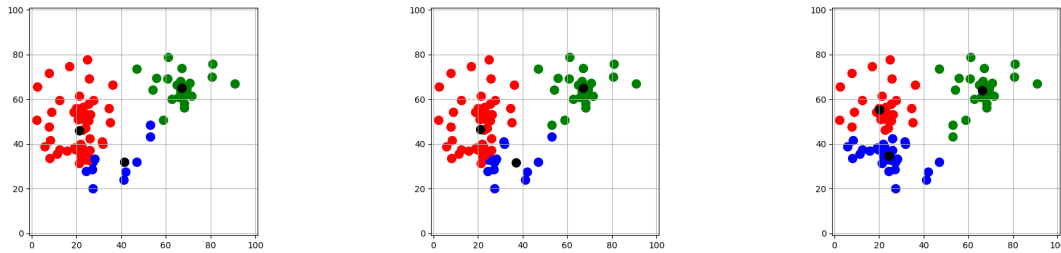
6. Itérer les barycentres.

Voici comment regrouper des points en les coloriant par groupe. Sur la figure de gauche on a tiré des points (presque) au hasard (on devine à peine 3 groupes). Sur la figure de droite, l'algorithme a séparé les points en 3 groupes pertinents (avec en noir les 3 barycentres que l'on va déterminer).



Voici les étapes graphiques de l'algorithme. Sur la figure de gauche ci-dessous, on part de trois points (en noir) qui jouent le rôle de barycentres initiaux (on les choisit au hasard ou bien trois points assez écartés de la fenêtre). On colorie les autres points selon le barycentre le plus proche. Étape suivante (figure du milieu) on calcule le nouveau barycentre pour les points rouges, le nouveau barycentre pour les points bleus... mais alors on doit aussi recalculer la couleur de tous les autres points (puisque les coordonnées des barycentres ont changé). On itère le processus (autres figures) jusqu'à ce que les couleurs des points ne changent plus.



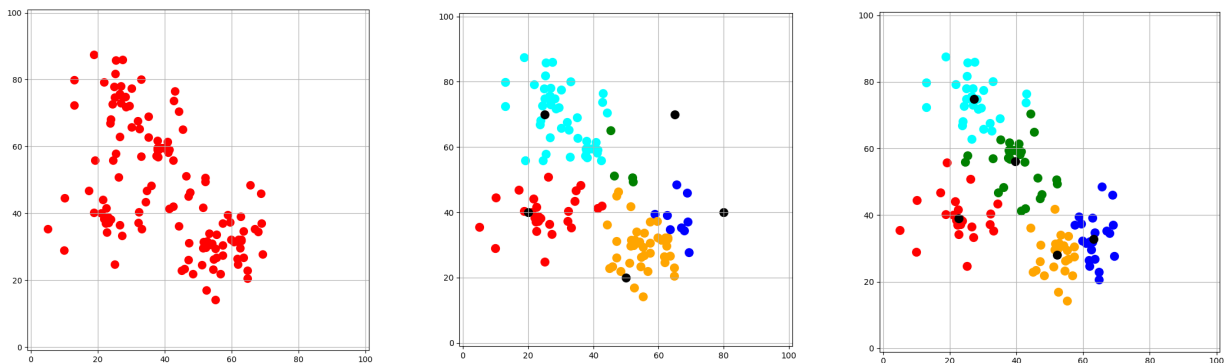


Programme une fonction `iterer_barycentres(points, barycentres_init)` qui effectue cette tâche.

Indications.

- Il s'agit juste, à chaque étape, d'utiliser la fonction `recalculer_barycentres()` suivie de `couleurs_barycentres_proches()` avec les nouvelles coordonnées des barycentres.
- On itère ces étapes jusqu'à ce que la liste des couleurs se stabilise (ou bien après un nombre fixé d'étapes).
- Pour remplacer la liste des anciens barycentres par la liste des nouveaux, utilise :
`barycentres = list(nouv_barycentres)`

On termine par un exemple avec 5 groupes de points (à gauche les points de départs, au centre les 5 barycentres initiaux et le premier coloriage, à droite le résultat après plusieurs itérations). Il faut parfois tester différentes configurations des barycentres initiaux pour que cela fonctionne bien.



Note.

Il y a une différence fondamentale entre cette activité et l'activité sur les k voisins. Il s'agit ici d'un *apprentissage non supervisé* c'est-à-dire qu'on laisse l'algorithme trouver tout seul les groupes. L'activité sur les k voisins est celle d'un *apprentissage supervisé* : certains points sont déjà coloriés rouge ou bleu et servent de référence, ensuite il s'agit de colorier les autres points suivant ce modèle.

Cours 4 (Neurone).

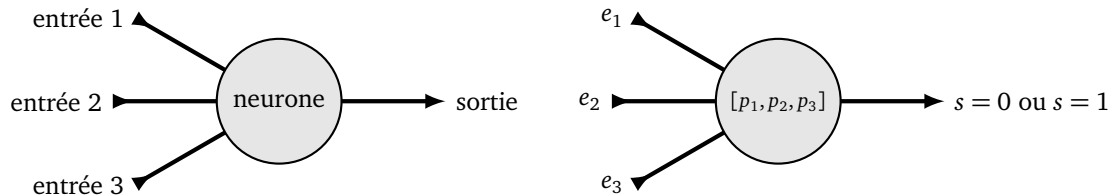
Des neurones.

Le cerveau est composé d'environ 100 milliards de neurones. Les réseaux de neurones artificiels s'inspirent du cerveau pour répondre à des problèmes complexes. Un réseau de neurones possède des coefficients qui sont calculés par un apprentissage. Le réseau est ensuite capable de répondre au problème posé, y compris lors de situations qui n'ont pas été rencontrées lors de l'apprentissage.

Un neurone.

Nous allons travailler avec un seul neurone. Notre modèle du neurone s'appelle le *perceptron*. Notre neurone fonctionne ainsi :

- Il reçoit en entrée des nombres, pour nous ce sera 3 réels e_1, e_2, e_3 .
- En sortie il renvoie un entier s qui vaut 0 ou 1. Si la sortie vaut 1, on dit que le neurone est *activé*.
- Chaque neurone est déterminé par des nombres, pour nous 3 réels p_1, p_2, p_3 . C'est *l'état* du neurone.



Activation du neurone.

Comment savoir si le neurone est activé? En fonction de l'entrée (e_1, e_2, e_3) et de l'état du neurone (p_1, p_2, p_3) on commence par calculer une valeur q selon la formule :

$$q = p_1 e_1 + p_2 e_2 + p_3 e_3.$$

Ensuite on regarde si q est plus grand ou plus petit que 1 pour déterminer si le neurone est activé (c'est-à-dire la valeur s renvoyée par le neurone) :

$$\begin{cases} s = 0 & \text{si } q < 1, \\ s = 1 & \text{si } q \geq 1. \end{cases}$$

Note bien qu'il n'y a que deux sorties possibles $s = 0$ ou $s = 1$.

Objectif.

Quelle tâche demande-t-on à notre neurone? Le neurone prend en entrée des nombres (e_1, e_2, e_3) et renvoie en sortie 0 ou 1. Il répond donc à une question du type « oui ou non? ». Plus précisément la question à laquelle répond le neurone est « ce point de coordonnées (e_1, e_2, e_3) est-il au-dessus ou en dessous de ce plan \mathcal{P} ? » Le plan \mathcal{P} dont il est question est le plan d'équation $p_1 x + p_2 y + p_3 z = 1$ déterminé par l'état (p_1, p_2, p_3) du neurone. On verra dans l'activité suivante comment cela permet de répondre à la question « cette couleur est-elle rouge ou pas? ».

Le problème principal est, qu'au départ, on ne connaît pas le plan \mathcal{P} qui répond au problème que l'on se pose, autrement dit on ne sait pas quel état (p_1, p_2, p_3) convient. Pour notre exemple cela signifie que l'ordinateur ne sait pas ce qu'est la couleur rouge. Il va donc falloir lui apprendre !

Apprentissage.

On part d'un état initial quelconque par exemple $(p_1, p_2, p_3) = (1, 1, 1)$. Pour faire évoluer l'état du neurone jusqu'à la bonne valeur de (p_1, p_2, p_3) on va l'entraîner comme un enfant : on lui montre une couleur et on lui dit « c'est du rouge », puis on lui montre une autre couleur et on lui dit « ce n'est pas du rouge ». À chaque étape l'état du neurone (p_1, p_2, p_3) est modifié.

Voici une étape d'entraînement : on donne une entrée (e_1, e_2, e_3) et l'objectif $b = 0$ ou $b = 1$ qui est la sortie attendue si le neurone était parfaitement paramétré. On calcule la sortie $s = 0$ ou $s = 1$ que renvoie le neurone selon l'entrée (e_1, e_2, e_3) dans son état actuel, ensuite plusieurs cas sont possibles :

- Si l'objectif b est égal à la sortie s alors le neurone fonctionne bien pour cette entrée, on ne change pas l'état du neurone.
- Si la sortie calculée est $s = 0$ alors que l'objectif est $b = 1$, alors on change l'état du neurone (p_1, p_2, p_3)

en un nouvel état (p'_1, p'_2, p'_3) selon la formule :

$$\begin{cases} p'_1 &= p_1 + \epsilon e_1 \\ p'_2 &= p_2 + \epsilon e_2 \\ p'_3 &= p_3 + \epsilon e_3 \end{cases}$$

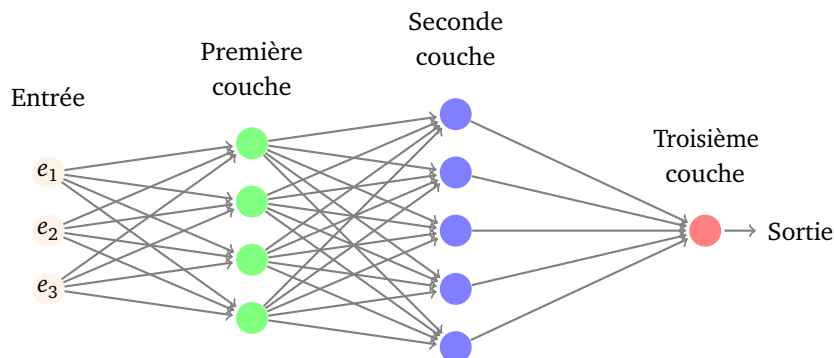
- Si la sortie calculée est $s = 1$ alors que l'objectif est $b = 0$, alors on change l'état du neurone selon la formule :

$$\begin{cases} p'_1 &= p_1 - \epsilon e_1 \\ p'_2 &= p_2 - \epsilon e_2 \\ p'_3 &= p_3 - \epsilon e_3 \end{cases}$$

Le paramètre ϵ est un petit réel. On prendra $\epsilon = 0.2$ par exemple. On répète ces étapes avec plusieurs entrées et objectifs. Géométriquement chaque apprentissage déplace un petit peu le plan \mathcal{P} pour mieux répondre au problème. L'état du neurone va converger vers un état (p_1, p_2, p_3) . Une fois la phase d'apprentissage terminée on conserve cet état final (p_1, p_2, p_3) . Maintenant on laisse répondre le neurone en regardant s'il s'active ou pas selon des entrées quelconques (e_1, e_2, e_3) (même si ces entrées ne font pas partie de la liste d'apprentissage).

Réseau de neurones.










Dans un réseau de neurones il y a plusieurs neurones organisés en couches. Les sorties d'une couche servent d'entrées pour la couche suivante. Un réseau de neurones simple et bien entraîné peut reconnaître des chiffres manuscrits (c'est un 0, c'est un 1...), des réseaux plus sophistiqués reconnaissent des photos (c'est un chat ou c'est un chien), jouent aux échecs (le meilleur coup est la reine en d7) et conduisent des voitures !



Cours 5 (Couleurs).

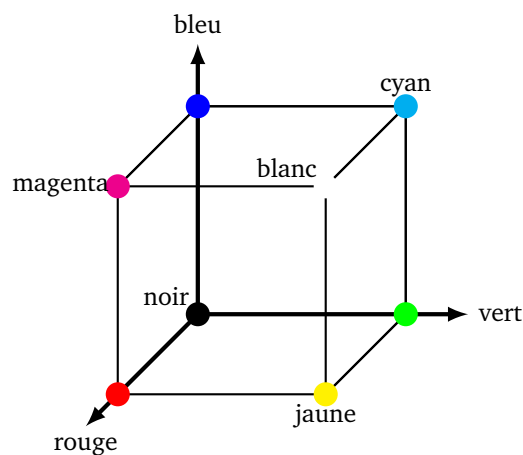
Code couleur *rgb*. Dans le système *rgb* on code une couleur selon ses niveaux de rouge/vert/bleu (*red/green/blue*). Donc une couleur est codée par trois réels (r, g, b) chacun allant de 0 à 1.

Voici quelques exemples de couleurs.

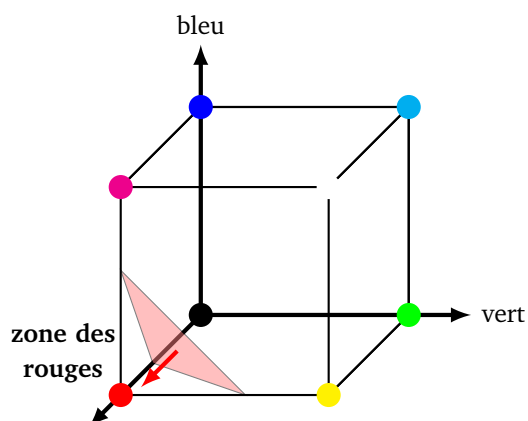
Couleur	Nom	Niveau de rouge	Niveau de vert	Niveau de bleu
	rouge	1	0	0
	vert	0	1	0
	bleu	0	0	1
	blanc	1	1	1
	noir	0	0	0
	orange	1	0.5	0
	gris	0.5	0.5	0.5
	cyan	1	1	0
	une nuance de rouge	0.8	0.2	0.1

Nuances de rouge. Dans l'usage la couleur rouge ne se limite pas au code (1, 0, 0), il peut y avoir du rouge clair, du rouge foncé, du rouge orangé... Par exemple le code (0.8, 0.2, 0.1) est bien une nuance de rouge (voir le tableau ci-dessus).

Si on considère l'espace de toutes les couleurs comme un cube (axe x pour le rouge variant de 0 à 1, axe y pour le vert, axe z pour le bleu) alors le rouge et ses nuances correspondent à une zone au voisinage du point (1, 0, 0).



Dans l'activité suivante nous allons programmer un neurone afin qu'il détecte une zone de rouge. Cette zone sera une portion du cube des couleurs coupé par un plan d'équation $p_1x + p_2y + p_3z = 1$ (où (x, y, z) jouent le rôle de (r, g, b)). Par apprentissage, on va déterminer les coefficients (p_1, p_2, p_3) qui conviennent.





Activité 4 (Neurone).

Objectifs : programmer un neurone qui après un apprentissage détecte si une couleur donnée est rouge ou pas.

1. Activation d'un neurone.

Programme une fonction `activation(neurone,entree)` qui selon l'état de `neurone = [p1,p2,p3]` et les valeurs `entree = [e1,e2,e3]` renvoie $s = 1$ en cas d'activation et $s = 0$ sinon. On rappelle que cette activation est déterminée par :

$$\begin{cases} s = 1 & \text{si } p_1e_1 + p_2e_2 + p_3e_3 \geq 1, \\ s = 0 & \text{sinon.} \end{cases}$$

Exemples.

- Avec `neurone = [1,2,3]` et `entree = [0.5,0,0]` alors on calcule $q = p_1e_1 + p_2e_2 + p_3e_3 = 1 \times 0.5 + 2 \times 0 + 3 \times 0 = 0.5 < 1$. Donc le neurone ne s'active pas, la sortie vaut $s = 0$.
- Avec `neurone = [1,2,3]` et `entree = [0,1,0.5]` alors on calcule $q = p_1e_1 + p_2e_2 + p_3e_3 = 1 \times 0 + 2 \times 1 + 3 \times 0.5 = 3.5 \geq 1$. Cette fois le neurone s'active et la sortie vaut $s = 1$.
- Pour `neurone = [1,0.5,2]` et `entree = [0.2,0.1,0.1]` est-ce que le neurone s'active ou pas ? Et avec `entree = [0.3,0.2,0.7]` ?

2. Apprentissage.

Programme une fonction `apprentissage(neurone,entree,objectif)` qui renvoie l'état $[p'_1, p'_2, p'_3]$ modifié du neurone après apprentissage avec l'entrée et l'objectif (0 ou 1) donné.

Voici comment faire :

- On note l'état actuel du neurone par $[p_1, p_2, p_3]$ et l'entrée donnée par $[e_1, e_2, e_3]$.
- On commence par calculer si le neurone s'active ou pas avec cette entrée. On note $s = 0$ ou $s = 1$ cette sortie.
- Si la sortie s est égale à l'objectif b à atteindre alors on conserve le neurone tel quel : $[p'_1, p'_2, p'_3] = [p_1, p_2, p_3]$. En effet le neurone a déjà le bon comportement pour cette entrée, on ne change donc rien.
- Si la sortie s est différente de l'objectif, alors on modifie l'état du neurone en $[p'_1, p'_2, p'_3]$ selon la formule suivante :

$$\begin{cases} p'_1 &= p_1 \pm \epsilon e_1 \\ p'_2 &= p_2 \pm \epsilon e_2 \\ p'_3 &= p_3 \pm \epsilon e_3 \end{cases}$$

On prendra $\epsilon = 0.2$ par exemple. Le signe est « + » si l'activation calculée est $s = 0$ alors que l'objectif est $b = 1$. Le signe est « - » si l'activation calculée est $s = 1$ alors que l'objectif est $b = 0$.

- La fonction renvoie $[p'_1, p'_2, p'_3]$.

Exemples. Partons du neurone dont l'état de départ `neurone` est $[p_1, p_2, p_3] = [1, 1, 1]$.

- Si l'entrée est $[e_1, e_2, e_3] = [1, 0, 2]$ et l'objectif à atteindre est $b = 1$, alors on calcule $s = 1$. Comme $s = b$ alors on ne fait rien et $[p'_1, p'_2, p'_3] = [p_1, p_2, p_3]$.
- Ce serait pareil si l'entrée était $[0.5, 0.1, 0.2]$ avec un objectif de $b = 0$ (car on a aussi $s = 0$).
- Si l'entrée est $[e_1, e_2, e_3] = [0.5, 0.2, 0]$ et l'objectif est $b = 1$, alors on calcule $s = 0$. Comme l'objectif est différent de la sortie, on modifie l'état du neurone, selon la formule $p'_i = p_i + \epsilon e_i$. On trouve donc :

$$p'_1 = 1 + 0.2 \times 0.5 = 1.1 \quad p'_2 = 1 + 0.2 \times 0.2 = 1.04 \quad p'_3 = 1 + 0.2 \times 0 = 1$$

Le nouvel état du neurone est donc $[1.1, 1.04, 1]$.

- On repart de l'état initial du neurone $[p_1, p_2, p_3] = [1, 1, 1]$. Si l'entrée est $[e_1, e_2, e_3] = [0, 1, 1]$ et l'objectif est $b = 0$, alors on calcule $s = 1$. Comme l'objectif est différent de la sortie on modifie l'état du neurone, selon la formule $p'_i = p_i - \epsilon e_i$. On trouve donc :

$$p'_1 = 1 - 0.2 \times 0 = 1 \quad p'_2 = 1 - 0.2 \times 1 = 0.8 \quad p'_3 = 1 - 0.2 \times 1 = 0.8$$

Indications. Pour éviter les problèmes en modifiant une liste tu peux commencer par la copier, avec par exemple :

```
nouv_neurone = list(neurone)
```

Tu peux ensuite modifier la liste `nouv_neurone`.

3. Apprentissage itéré.

Pour que le neurone s'entraîne il faut lui procurer plusieurs données. Programme une fonction `epoque_apprentissage(neurone_init, liste_entrees_objectifs)` qui calcule le nouvel état du neurone après entraînement sur chaque élément de la liste. La liste d'entraînement est une liste composée de couples entrée/objectif attendu :

```
[ ([1,0,0],1), ([0,1,1],0), ([0,1,0],0), ... ]
```

Il s'agit juste d'itérer la fonction `apprentissage()` sur chaque élément de la liste. Un entraînement sur la totalité du jeu de tests s'appelle une *époque*.

4. Apprentissage de la couleur rouge.

Entraîne ton neurone afin qu'il reconnaisse la couleur rouge.








Méthode.









- En entrée les paramètres $[e_1, e_2, e_3]$ correspondent au code couleur $[r, g, b]$ (trois nombres réels entre 0 et 1).
- Si la sortie vaut 1 alors le neurone déclare que c'est du rouge, si la sortie vaut 0 il déclare que ce n'est pas du rouge.
- Pars d'un neurone dans un état quelconque, par exemple $[p_1, p_2, p_3] = [1, 1, 1]$ et fait le évoluer par apprentissage sur toute la liste d'entraînement (première époque).
- À partir du nouvel état du neurone, recommence l'apprentissage avec toujours la même liste d'entraînement (deuxième époque, puis troisième époque...).
- Au bout d'une dizaine d'époques l'état du neurone devrait être stable et bien paramétré.

Entraînement. Voici le couple entrée/objectif que tu peux utiliser pour apprendre la couleur rouge.

```
liste_entrees_objectifs = [
    ([1,0,0],1), ([0,1,1],0), ([1,1,0],0),
    ([1,0,0.2],1), ([0,1,0],0), ([0,0,0],0),
    ([1,0,1],0), ([0.7,0,0],1), ([0.5,0.5,0.5],0),
    ([0.9,0.2,0],1), ([0.9,0,0],1), ([1,1,1],0),
    ([0.2,1,0],0), ([0.8,0.2,0],1), ([0.7,0.1,0.1],1) ]
```

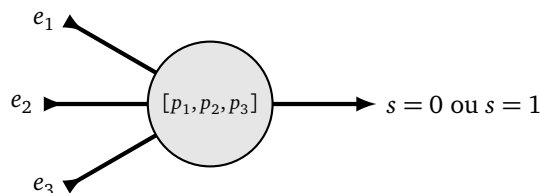
Par exemple le couple $([1, 0, 0], 1)$ signifie que la couleur de code *rgb* $[1, 0, 0]$ est du rouge ; le couple $([0, 1, 1], 0)$ signifie que la couleur de code *rgb* $[0, 1, 1]$ n'est pas du rouge. Note que pour faire comprendre au neurone ce qu'est du rouge, il faut aussi lui montrer des couleurs différentes du rouge. Voici la liste d'apprentissage sous la forme d'un tableau de couleur.

Couleur	Code <i>rgb</i>	Est-ce du rouge ?
	(1, 0, 0)	oui
	(1, 0, 0.2)	oui
	(0.7, 0, 0)	oui
	(0.9, 0.2, 0)	oui
	(0.8, 0.2, 0)	oui
	(0.9, 0, 0)	oui
	(0.7, 0.1, 0.1)	oui






Couleur	Code <i>rgb</i>	Est-ce du rouge ?
	(0, 1, 1)	non
	(1, 1, 0)	non
	(0, 1, 0)	non
	(0, 0, 0)	non
	(1, 0, 1)	non
	(0.5, 0.5, 0.5)	non
	(1, 1, 1)	non
	(0.2, 1, 0)	non

Liste d'entraînement de la couleur rouge.

Résultats. En partant de l'état initial $[p_1, p_2, p_3] = [1, 1, 1]$ avec un pas de $\epsilon = 0.2$ alors l'état du neurone converge en 10 époques vers $[1.66, -0.78, -0.66]$ (ces valeurs ne sont pas uniques, elles dépendent de l'échantillon d'apprentissage, de ϵ et du nombre d'époques).



Vérifications. Vérifions que notre neurone, avec l'état $[p_1, p_2, p_3] = [1.66, -0.78, -0.66]$, détecte correctement le rouge, c'est-à-dire que la fonction activation() renvoie 1 uniquement pour une couleur rouge en entrée. Voici les résultats obtenus :

Couleur	Code <i>rgb</i>	$q = p_1 e_1 + p_2 e_2 + p_3 e_3$	Sortie <i>s</i>	Rouge ?
	(0.9, 0, 0)	1.49	1	oui
	(1, 0.2, 0.2)	1.37	1	oui
	(0, 0, 1)	-0.66	0	non
	(1, 0.5, 0)	1.27	1	oui
	(0.7, 0.5, 0.4)	0.50	0	non

Détection du rouge par le neurone.

Notes.

- Notre neurone fonctionne remarquablement bien !
- Cependant il considère le orange comme du rouge (les goûts et les couleurs...).
- Remarque aussi que l'on n'a jamais montré au neurone la couleur bleue lors de l'apprentissage, mais au final il sait que le bleu n'est pas du rouge !

Projet. Programme une interface graphique qui permet l'apprentissage interactif en proposant à l'utilisateur des couleurs avec un choix « rouge/pas rouge » ; affiche aussi l'état du neurone, son évolution et sa réussite sur un jeu de tests.