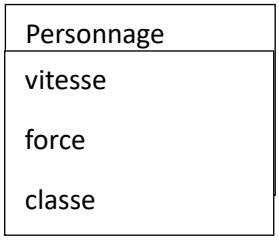
Partie 1 Des Elfes et de Trolls

1. Question 1

On désire modéliser des catégories de personnage pour un jeu vidéo. On utilise le formalisme de la POO sans notion d'héritage.

Chaque personnage sera modélisé par un objet de type personnage avec trois attributs : sa vitesse, sa force et sa classe 1 pour un Troll et 2 pour un Elfe

1.1. Ecrire le code de la classe Personnage avec un constructeur paramétré qui permet lors de la création du personnage de donner une valeur aux trois attributs




```
class Personnage():
    def __init__(self, vitesse,force, classe):
        self.force = force
        self.vitesse =  vitesse
        self.classe = classe
```

1.2. Ecrire le code de la méthode __str__ qui permet de « printer » les personnages crées comme l'exemple ci-dessous

```
troll1 = Personnage(7,35,1)
elfe1 = Personnage(15,12,2)
```

Affiche

```
>>> print(troll1)
Je suis un Troll je possède  une vitesse de : 7 et une force de 35

>>> print(elfe1)
Je suis un Elfe je possède  une vitesse de : 15 et une force de 12
```

```
def __str__(self) :
        if self.classe  == 1 :
            type = "Troll"
        else :
            type = "Elfe"
        return(f"Je suis un {type} je possède  une vitesse de : {self.vitesse} et une force
de {self.force}")
```

1.3 Les Personnages peuvent utiliser une potion magique qui dans le cas d'un Troll augmente sa vitesse de 10 et dans le cas d'un Elfe sa force de 10 aussi. Ecrire la méthode **potion_magique** qui modifie leur caractéristique appliquer-la à un Elfe

```
>>> print(elfe1)
Je suis un Elfe je possède  une vitesse de : 15 et une force de 22
```

```
def potion_magique(self):
        if self.classe  == 1 :
            self.vitesse += 10
        else :
            self.force += 10
```

`troll1 = Personnage(7,35,1)`
`elfe1 = Personnage(15,12,2)`
`print(troll1)`
`print(elfe1)`
`elfe1.potion_magique()`
`troll1.potion_magique()`
`print(troll1)`
`print(elfe1)`
	Je suis un Troll je possède une vitesse de : 7 et une force de 35
	
	Je suis un Elfe je possède une vitesse de : 15 et une force de 12
	
	Je suis un Troll je possède une vitesse de : 17 et une force de 35
	
	Je suis un Elfe je possède une vitesse de : 15 et une force de 22

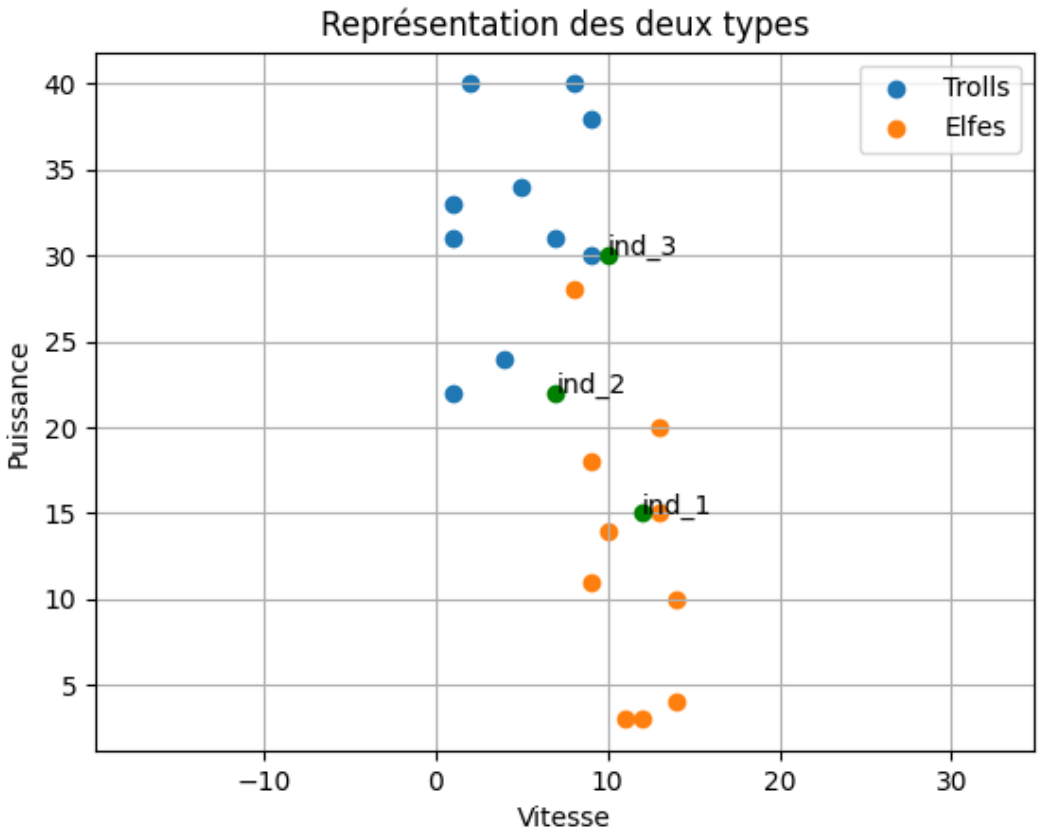
2. Question 2

Supposons que des personnages d'un jeu aient deux caractéristiques principales : la vitesse et la puissance.

Parmi les différentes classes de personnages, intéressons-nous à deux d'entre-elles :

- les Trolls, qui sont lents mais très puissants ;
- les Elfes, qui sont rapides mais moins puissants que les Trolls.

Ici il y a deux **classes** (Troll ou Elfe) et deux **caractéristiques** (vitesse et puissance).



L'axe des x vitesse

L'axe des y la force

2.1. Question Trois nouveaux individus apparaissent : **ind_1** vitesse 12 , puissance 15 ; **ind_2** vitesse 7, puissance 22 ; **ind_3** vitesse 10 puissance 30. Positionner ces individus sur le graphique et indiquer à quelle classe ils appartiennent. Voir ci-dessus

Les caractéristiques de l'échantillon des Personnages ont été extraits dans une liste de listes :

personnages = [[14, 10, 2], [9, 11, 2], [10, 14, 2], [8, 28, 2], [1, 31, 1], [14, 4, 2], [13, 20, 2], [4, 24, 1], [5, 34, 1], [1, 22, 1], [9, 30, 1], [13, 15, 2], [8, 40, 1], [9, 18, 2], [12, 3, 2], [2, 40, 1], [11, 3, 2], [7, 31, 1], [1, 33, 1], [9, 38, 1]]

ainsi chaque sous-liste donne des caractéristiques d'un personnage :

* le premier nombre est sa vitesse ;

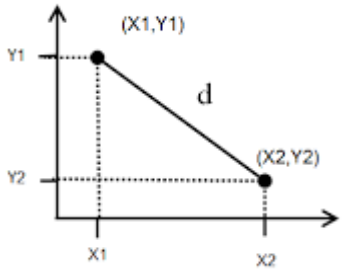
* le deuxième nombre est sa puissance ;

* le troisième nombre est sa classe : 1 pour un Troll et 2 pour un Elfe.

2.2. Question

La distance euclidienne est choisie pour quantifier la proximité des personnages avec **l'ind_2 (7,22, None).**

On rappelle d :



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

from math import sqrt, pow

racine carrée

sqrt(100) # retourne 10

L'axe des x vitesse

L'axe des y la force

Calculer la distance euclidienne entre ind_2 et les deux premiers individus de la liste : [14, 10, 2] et [9, 11, 2],

$$d_0 = \sqrt{(14-7)^2 + (10-22)^2} = 13.89$$

$$d_1 = \sqrt{(9-7)^2 + (11-22)^2} = 11.18$$

Donner le code de la fonction **distance** qui retourne la distance euclidienne entre deux personnages perso1 et perso2 dans le repère y force, x vitesse

```
def distance (perso1,perso2):
    return sqrt(pow((perso1[0]-perso2[0]),2)+pow((perso1[1]-perso2[1]),2))
```

ou

```
def distance_bis(perso1,perso2) :

    return sqrt((perso1[0]-perso2[0])**2+(perso1[1]-perso2[1])**2)
```

2.3. Question

Donner le code de la fonction **ajoute_distance** qui prend comme argument une liste de personnage et le personnage ind_2 pour ajouter à tous les personnes de la liste un quatrième élément qui est la distance entre ind2 et chaque personnage :

```
ajoute_distances(personnages,ind_2)
```

On obtient

Personnages : [[14, 10, 2, 13.892443989449804], [9, 11, 2, 11.180339887498949], [10, 14, 2, 8.54400374531753], [8, 28, 2, 6.082762530298219], [1, 31, 1, 10.816653826391969], [14, 4, 2, 19.313207915827967], [13, 20, 2, 6.324555320336759], [4, 24, 1, 3.605551275463989], [5, 34, 1, 12.165525060596439], [1, 22, 1, 6.0], [9, 30, 1, 8.246211251235321], [13, 15, 2, 9.219544457292887], [8, 40, 1, 18.027756377319946], [9, 18, 2, 4.47213595499958], [12, 3, 2, 19.6468827043885], [2, 40, 1, 18.681541692269406], [11, 3, 2, 19.4164878389476], [7, 31, 1, 9.0], [1, 33, 1, 12.529964086141668], [9, 38, 1, 16.1245154965971]]

```
def ajoute_distances(l,ind_2):
    for i in range(len(l)):
        l[i].append(distance_bis(l[i],ind_2))
```

2.4. Question

Compléter le code de la fonction tri_sélection qui trie la liste précédente avec les distances qui la trie en fonction de ces distances :

```
def tri_selection(liste):
    """
    Modifie l'ordre de la liste en mettant dans l'ordre tous les éléments en fonction de la distance dans l'ordre croissant
    """
    for i in range(len(liste)):
        min = i
        for j in range(i+1, len(liste)):
            if liste[j][3] < liste[min][3]:
                min = j
        liste[i], liste[min] = liste[min], liste[i]
```

On obtient

```
>>> personnages
[[4, 24, 1, 3.605551275463989], [9, 18, 2, 4.47213595499958], [1, 22, 1, 6.0], [8, 28, 2, 6.082762530298219], [13, 20, 2, 6.324555320336759], [9, 30, 1, 8.246211251235321], [10, 14, 2, 8.54400374531753], [7, 31, 1, 9.0], [13, 15, 2, 9.219544457292887], [1, 31, 1, 10.816653826391969], [9, 11, 2, 11.180339887498949], [5, 34, 1, 12.165525060596439], [1, 33, 1, 12.529964086141668], [14, 10, 2, 13.892443989449804], [9, 38, 1, 16.1245154965971], [8, 40, 1, 18.027756377319946], [2, 40, 1, 18.681541692269406], [14, 4, 2, 19.313207915827967], [11, 3, 2, 19.4164878389476], [12, 3, 2, 19.6468827043885]]
```

2.5. Question

Compléter le code de la fonction détermine qui prend comme arguments la liste précédente et r un nombre k et qui renvoie le type d'individu en fonction des k plus proches voisins.

```
def determine(liste, k):
    trolls = 0
    elfes = 0

    for i in range(k):
        if liste[i][2] == 1 :
            trolls +=1
        else :
            elfes += 1

    if elfes == trolls :
        print("Indéterminé")
    elif trolls > elfes :
        print("Trolls")
    else :
        print("Elfes")
```

Affiche

```
>>> determine(personnages,5)
Elfes
```

Partie 2 La recherche dichotomique (extrait d'un sujet de bac)

3. Question

3.1. La recherche d'un élément dans un tableau avec une méthode dichotomique ne peut se faire que si le tableau est trié.

a) Vrai

b) Faux

3.2. Le coût d'un algorithme de recherche dichotomique est :

a) Constant : Complexité O(1)

b) Linéaire : Complexité O(n)

c) Logarithmique : Complexité O(log(n))

3.3. Justifier pourquoi l'entier fin – deb est un variant de boucle qui montre la terminaison du programme de recherche dichotomique suivant ;

```
def rechercheDicho (elem, liste):
"""
Cette fonction indique si un élément se trouve dans un
tableau.
Elle utilise la méthode de recherche dichotomique.
Elle prend en arguments :
- elem : élément à rechercher de type string
- liste : liste d'éléments de type string triée
par ordre croissant
Elle renvoie un booléen correspondant à la présence ou
non de l'élément
"""
    deb = 0
    fin = len(liste)-1
    m = (deb+fin)//2
    while deb <= fin :
        if liste[m] == elem :
            return True , m
        elif liste[m] > elem :
            fin = m-1
        else :
            deb = m+1
        m = (deb+fin)//2
    return False ,-1
```

Q3,6

La condition de boucle étant début <= fin , cela correspond exactement à ce que notre variant soit positif ou nul. Montrons maintenant que le variant décroît strictement lors de l'exécution du corps de la boucle.

On commence par définir milieu = (début + fin) // 2.

En particulier, on a alors début <= milieu<= fin.

Ensuite, trois cas sont possibles.

- si liste[milieu] == val, on sort directement de la boucle à l'aide d'un return. La terminaison est assurée.
- si liste[milieu] > val, on modifie la valeur de fin. En appelant fin2 cette nouvelle valeur, on a : fin2 - début < milieu - début <= fin - début car fin2 = milieu - 1 < milieu.
Ainsi, le variant a strictement décru.
- sinon, on modifie début et on a de même :
fin -debut2 <fin - milieu <= fin - début

De même, le variant a strictement décru.

Donc soit on trouve milieu qui correspond à la valeur cherchée soit on ne le trouve pas et comme le variant décroit on arrivera à ce que la condition début <= fin ne soit plus vraie.

Le programme de recherche dichotomique de l'annexe 1 de l'exercice 2 est utilisé pour effectuer des recherches dans une liste.

Dans l'ensemble de cette partie, on considère la liste : Lnoms = ["alice", "bob","etienne", "hector", "lea", "nathan", "paul"].

3.4. Expliquer pourquoi en ligne 2, on a «fin = len(liste)-1» plutôt que «fin = 6».

fin = len(liste) -1 généralise le fonctionnement de la fonction pour toute taille de liste . fin = 6 limite le bon foisonnement de la fonction aux listes de taille 6. C'est d'autant plus pertinent que l'on ne connait pas la taille de la liste passée en paramètre.

3.5. Donner la trace complète de l'exécution rechercheDicho("lea", Lnoms) en complétant le tableau ci-dessous sur votre copie :

Variables			Condition	Valeur renvoyée
deb	**Fin**	**m**	**deb <= fin**	
0	6	3	true	
4	6	5	true	
4	4	4	false	True

3.6. Sur votre copie, modifier le code du corps de la fonction rechercheDicho() pour qu'elle renvoie aussi la position (indice) de l'élément cherché ou -1 si l'élément n'est pas trouvé. On pourra indiquer sur la copie le numéro des lignes modifiées, à supprimer ou à insérer s'il y a lieu.