1. Recherche dichotomique
   a) Soit a = [1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20]
   Appliquer « à la main » la méthode de recherche dichotomique pour savoir si le nombre 14 est dans la liste, puis le nombre 2.
   b) Ecrire une fonction dichotomie(liste, valeur, mode="") qui recherche l'élément valeur dans la liste et qui renvoie un booléen : *True* si la valeur est présente, *False* sinon.
      - mode = "n" : permet d'afficher le nombre de tests effectuées à la fin.
      - mode = "d" : permet d'afficher le détail du debug avec l'évolution de la recherche étape par étape.

2. Le jeu du nombre mystère
   a) On reprend le jeu du nombre mystère vu dans le TP_10. Cette fois-ci, c'est l'utilisateur qui choisit un nombre mystère entre 0 et 100 et c'est l'ordinateur qui doit le retrouver avec comme seules indications « Trop petit », « Trop grand » ou « Bravo, vous avez trouvé ! » (On pourra se contenter de "+", "-" ou "="). Indiquer le nombre d'essais utilisés par l'ordinateur pour trouver le nombre.
   b) Quel est le plus grand nombre d'essais que l'ordinateur peut avoir à utiliser si l'on choisit bien le nombre mystère ? Quels sont ces nombres mystères plus difficiles à trouver ?

3. Algorithme glouton : Le rendu de monnaie
   Etant donné une liste de valeurs de pièces (ou billets) possibles, comment rendre une somme d'argent donnée de façon optimale, c'est-à-dire en utilisant le nombre minimal de pièces (ou billets) ?
   Exemple : On considère que l'on dispose de pièces (ou billets) de : 1 €, 2 €, 5€, 10 €, 20 €, 50 €, 100 €, 200 € et 500 €.
   a) Trouver « à la main » une méthode pour rendre de manière optimale les sommes suivantes : 304 €, 485 € et 893 €.
   b) Ecrire une fonction monnaie(somme, liste_valeurs) qui renvoie une liste des pièces à rendre en en utilisant le moins possible.
   c) Tester cette fonction en considérant cette fois-ci que l'on dispose de pièces de : 1 €, 6 € et 10 € pour un rendu de monnaie sur la somme de 38 €… Peut-on faire mieux ?
   d) Que répond la fonction si que l'on dispose cette fois-ci de pièces de : 3 €, 6 € et 10 € pour un rendu de monnaie sur la somme de 38 € ? Qu'en pensez-vous ?
   e) Que répond la fonction si que l'on dispose cette fois-ci de pièces de : 0,10 €, 0,60 € et 1,00 € pour un rendu de monnaie sur la somme de 3,80 € ? Qu'en pensez-vous ?

4. Algorithme glouton : Le planning d'activités
   a) Vous organisez une journée de conférences dans votre lycée de 8h à 20h. Vous avez reçu la disponibilité de 10 conférenciers :
      - Mr A. : de 8h à 10h
      - Mme B. : de 8h à 11h
      - Mr C. : de 9h à 12h
      - Mme D. : de 10h à 13h
      - Mr E. : de 13h à 17h
      - Mr F. : 14h à 16h
      - Mme G. de 14h à 17h
      - Mme H. : de 16h à 18h
      - Mr I. : de 17h à 19h
      - Mme J. : de 18h à 20h

      Malheureusement, vous n'avez qu'une seule salle de conférence dans le lycée, quels conférenciers proposez-vous pour optimiser la journée avec le plus grand nombre de conférences organisées sans chevauchement ?
   b) Trouver un algorithme efficace pour automatiser cette tâche. Indication : Utiliser un tri astucieux…
   c) Ecrire une fonction planning(liste_conf) qui renvoie la liste des conférenciers à proposer, à partir d'une liste de leurs disponibilités contenant les noms et créneaux de chacun :
      [(nom_1, début_1, fin_1), (nom_2, début_2, fin_2), …]
   d) Tester la fonction avec l'exemple du a) puis avec la liste suivante :
   liste_conferenciers = [("A", 8, 10), ("B", 9, 11), ("C", 9, 12), ("D", 10, 12), ("E", 11, 14), ("F", 12, 15), ("G", 13, 14), ("H", 13, 16), ("I", 14, 15), ("J", 15, 17), ("K", 16, 18), ("L", 16, 19), ("M", 17, 18), ("N", 17, 19), ("O", 18, 20)]

5. Algorithme des k plus proches voisins : Les iris
   Le fichier TP18iris.csv contient un tableau répertoriant la largeur et la longueur des sépales et des pétales (en cm) de trois espèces différentes d'iris, correspondant à 50 fleurs collectées par Edgar Anderson et étudiées par Ronald Fisher en 1936.
   a) Reprendre les fonctions vues dans le TP_13 pour lire les données du fichier CSV en les stockant dans une liste de dictionnaires en Python.
   b) On observe sur les nuages de points, représentant les données deux par deux, visibles sur la page Wikipédia nommée « Iris de Fisher » qu'il est plus pertinent d'étudier en particulier la largeur et la longueur des pétales.
      Ecrire une fonction distance_petales(fleur_1, fleur_2) qui renvoie la distance euclidienne entre deux fleurs en ne tenant compte que de la largeur et la longueur des pétales.
      C'est-à-dire que si on a fleur_1 = {″sepal_length″ : sl_1, ″sepal_width″ : sw_1, ″petal_length″ : pl_1, ″petal_width″ : pw_1 , ″species″ : sp_1} et fleur_2 avec les valeurs sl_2, sw_2, pl_2, pw_2, sp_2, alors la distance vaut :
      $$d = \sqrt{(pl_2 - pl_1)^2 + (pw_2 - pw_1)^2}$$
   c) Ecrire une fonction voisins_petales(fleur_0, k) qui renvoie la liste des k plus proches voisins de la fleur_0 en utilisant la distance vue précédemment et les données du fichier CSV.
   d) Ecrire une fonction prediction_petales(fleur_0, k) qui va utiliser la fonction précédente pour prédire l'espèce d'une fleur_0 dont on ne connait que la largeur et la longueur des pétales en renvoyant un couple contenant : l'espèce majoritaire parmi les k plus proches voisins et un pourcentage de fiabilité égal au quotient du nombre d'occurrence de cette espèce sur le nombre de voisins en ne considérant que la taille des pétales.
   e) Tester la fonction pour prédire l'espèce des fleurs suivantes :
      fleur_1 : sl_1 = 5.0, sw_1 = 3.0, pl_1 = 0.5, pw_1 = 0.5,
      fleur_2 : sl_2 = 7.0, sw_2 = 3.0, pl_2 = 6.0, pw_2 = 2.0,
      fleur_3 : sl_1 = 5.5, sw_1 = 3.0, pl_1 = 2.5, pw_1 = 0.8,
      fleur_4 : sl_2 = 6.0, sw_2 = 2.5, pl_2 = 4.5, pw_2 = 1.6,
      fleur_5 : sl_3 = 4.8, sw_3 = 2.8, pl_3 = 5.2, pw_3 = 1.5
      On essaiera successivement avec $k = 1$, $k = 3$, $k = 5$, puis $k = 7$.
   f) Modifier la fonction de distance utilisée en prenant en compte les sépales, puis refaire les tests.