Exercice 1 : les impairs

Écrire une fonction qui prend 2 nombres en paramètres, et qui affiche, dans l'ordre croissant, tous les entiers impairs se trouvant entre ces deux nombres. Vous devez afficher ces nombres, en les séparant uniquement d'un espace.

Exemple : fonction(42.75, 52.23) doit renvoyer 43 45 47 49 51

Exercice 2 : le jeu du plus ou moins

Coder le jeu suivant.
Pour commencer l'ordinateur va choisir au hasard un nombre compris entre 1 et 100.
L'utilisateur doit alors deviner ce nombre comme ceci :
L'utilisateur propose un nombre. L'ordinateur lui dit s'il est trop petit ou trop grand, et ainsi de suite jusqu'à ce que l'utilisateur aie trouvé le bon nombre.
On pourra ajouter **en options** dans ce jeu les fonctionnalités suivantes :

- L'ordinateur affiche un message d'erreur si l'utilisateur propose un nombre inférieur à 0 ou supérieur à 100.
- Vous pouvez choisir le niveau de difficulté (c'est-à-dire la taille de l'intervalle possible)
- Vous pouvez inclure un mode multijoueur où un autre utilisateur entre le nombre à deviner
- Vous pouvez ajouter une variable "compteur" qui fait en sorte de dire à l'utilisateur en combien de coups il a réussi à découvrir le nombre (ex. : "Vous avez trouvé le nombre en 5 coups")
- Vous pouvez ajouter un second compteur qui somme les scores de chaque partie
- Et enfin vous pouvez demander à l'utilisateur s'il veut refaire une partie ou pas

Exercice 3 : les plus grands nombres

Générer une liste aléatoire de 100 entiers puis en extraire les 10 plus grands.
Généraliser votre algorithme en créant une fonction prenant comme paramètres la taille de la liste initiale et le nombre d'entiers à extraire.

Contrainte : ne pas utiliser les fonctions de tri sort() ou sorted(). Vous pouvez toutefois utiliser ces fonctions dans un second temps pour vérifier vos résultats.

Exercice 4 : les algorithmes de tri

Écrire une fonction permettant de trier une liste. Ne pas utiliser les méthodes intégrées `sort()` et `sorted()`.
Pour aller plus loin :

- utiliser au moins 3 algorithmes différents
- afficher les temps d'exécution des différents algorithmes à l'aide du package *time*
- comparer les au temps d'exécution de méthodes `sort()` et `sorted()`

Exercice 5 : des conversions

Écrire un programme qui convertit un nombre entier de secondes fourni au départ, en un nombre d'années, de mois, de jours, de minutes et de secondes.
Contraintes:

- Ne pas utiliser le package `datetime`
- Afficher le résultat sous la forme :

```
34300615967919352555 secondes correspondent à :
108 692 093 086 années 8 mois 1 jours
15 heures 51 minutes 28 secondes
```

Écrire un programme qui convertit en mètres par seconde et en km/h une vitesse fournie par l'utilisateur en miles/heure (1 mile = 1609 mètres). Afficher le résultat avec uniquement 2 chiffres après la virgule.

Exercice 6 : le triangle de Pascal

Soit le tableau de Pascal :

1ère ligne	1	1								
2ème ligne	1	2	1							
3ème ligne	1	3	3	1						
4ème ligne	1	4	6	4	1					
5ème ligne	1	5	10	10	5	1				
6ème ligne	1	6	15+	20	15	6	1			
7ème ligne	1	7	21	35	35	21	7	1		
8ème ligne	1	8	28	56	70	56	28	8	1	
9ème ligne	1	9	36	84	126	126	84	36	9	1

Chaque élément d'une ligne s'obtient en sommant les deux termes au-dessus et à gauche de l'élément. Les premiers et derniers termes
Exemple : ligne 7, on a 35 = 20 + 15.

Écrire une fonction qui prend comme paramètre le numéro de ligne n et renvoie la plus grande valeur de la ligne n du tableau de Pascal.

Exercice 7 : manipulations de dictionnaires

Écrire une fonction qui échange les clés et les valeurs d'un dictionnaire (ce qui permettra par exemple de transformer un dictionnaire anglais/français en un dictionnaire français/anglais). On suppose qu'il n'y a pas de valeurs en double pour simplifier.

Exercice 8 : quelques notes

Écrire un programme permettant d'entrer des notes d'élèves d'élèves sur 20 jusqu'à ce l'utilisateur saisisse une note vide. Construire une liste et à chaque nouvelle entrée, afficher le nombre de notes entrées, la note la plus élevée, la note la plus basse, la moyenne de toutes les notes.

Facultatif : utiliser le module *pickle* pour enregistrer la liste de note et la récupérer si nécessaire. À chaque fois que l'utilisateur commence, lui demander s'il veut continuer une des listes existantes, en créer une nouvelle ou écraser une de celles existantes.

Exercice 9 : une première base de données

Écrire un script qui crée un système de base de données fonctionnant à l'aide d'un dictionnaire, et dans lequel vous mémoriserez les noms d'une série d'individus, leur âge,

leur sexe et leur taille. Votre script devra comporter 4 fonctions : la première pour le remplissage du dictionnaire, la seconde pour sa consultation, la troisième pour sa sauvegarde au format texte et la quatrième pour reconstituer le dictionnaire à partir du fichier texte.
La fonction de remplissage doit permettre d'ajouter les informations des individus au dictionnaire : le nom de l'individu servira de clé d'accès, et les valeurs seront constituées de tuples (âge, sexe, taille), dans lesquels l'âge sera exprimé en années (donnée de type entier), le sexe au format 'F' ou 'M' et la taille en mètres (donnée de type réel).
La fonction de consultation doit permettre à l'utilisateur d'obtenir à partir d'un nom le triplet « âge, sexe, taille » correspondant. Le résultat de la requête devra être une ligne de texte bien formatée, telle par exemple : « Nom : Xabi - âge : 25 ans - sexe : M - taille : 1.74 m ».
La fonction de sauvegarde doit stocker dans chaque ligne de votre fichier texte un élément du dictionnaire. Elle sera formatée de manière à bien séparer la clé et les différentes valeurs.
Au début de chaque utilisation, demander à l'utilisateur ce qu'il souhaite faire (compléter ou consulter un dictionnaire)