Menu

Sujet 0 : niveau Moyen Le juste Prix
C'est un sujet généreusement guidé qui vous donne une bonne idée de la démarche de projet, que vous le choisissiez ou pas

Sujet 1 : niveau facile
Trouver le maximum entre deux valeurs, entre plusieurs valeurs, savoir compter le nombre de fois où il est proposé

Sujet 2 : niveau facile Honorin & Quentin L
Calculateur de moyenne Analyseur de moyenne

Sujet 3 : niveau facile (très guidé)
Ouverture et lecture d'un fichier avec Python

Sujet 4 : niveau moyen Alina & Lucas
Utilisation de turtle pour faire des dessins

Sujet 6 : niveau difficile
Travail avec les anagrammes

Sujet 7 : niveau moyen
La carte manquante

Sujet 8 : niveau facile : les aventures d'Hercule

Sujet 9 : niveau moyen Hector
Fonctions permettant de créer des grilles en mode texte

Sujet 10 : niveau difficile : les numéros pernicieux

Sujet 11 : niveau moyen
La plus longue séquence de Collatz La suite de Fibonacci

Sujet 12 : niveau moyen - Du romain au décimal Matthias & Quentin N.
Créer deux fonctions permettant de traduire des nombres en notation romaine en notation décimale (avec chiffres arabes) et vice versa

Sujet 13 : niveau moyen
Mélange à queue d'aronde Distance

Sujet 14 : niveau moyen **Morse** Enzo & Safaa
traducteurs en langage Morse vers alphabet classique et vice versa

Sujet 15 : niveau facile
Fonctions de géométrie analytique de base

Sujet 16 : niveau facile Iman & Yves Adrien
Logiciel d'apprentissage des tables de multiplication

Sujet 17 : niveau facile Thomas & Nael
Analyseur de mot de passe

Sujet 18 : niveau facile
Détection de palindromes

Sujet 19 : niveau facile
Le code de César

Sujet 20 : niveau facile
calculatrice virtuelle

Sujet 0 : niveau Moyen mais super guidé

Le Juste Prix

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet00.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

But du jeu

Le joueur doit deviner un prix (entier), il fait des propositions et à chaque proposition on lui dit « c'est plus » ou « c'est moins ». Il gagne lorsqu'il a trouvé le juste prix.

Vous allez donc écrire un programme Python :

- dans lequel l'ordinateur choisit au hasard un nombre entier entre 0 et 100 (qui correspond à un prix) ;
- demande à l'utilisateur de deviner ce prix en le saisissant au clavier ;
- indique à l'utilisateur si « c'est plus » ou « c'est moins » ;
- se termine lorsque l'utilisateur a deviné le juste prix.

Contrainte supplémentaire : Le jeu devra contenir deux modes de jeu :

- Mode facile : le joueur a autant de tentatives qu'il souhaite et le jeu doit s'arrêter lorsque le joueur a trouvé le nombre mystère.
- Mode difficile : le joueur n'a que 10 essais pour trouver le nombre mystère ; si le joueur trouve en moins de 10 essais il gagne et la partie s'arrête, sinon il perd.

Comme il s'agit du premier mini-projet, celui-ci sera un peu guidé, de manière à vous montrer la méthodologie de projet.

Concevoir le mode facile

Je vais vous guider dans la conception du premier mode de jeu.

Avant de commencer un projet, il est nécessaire de procéder par étapes :

1) Réfléchir à la structure du programme à écrire (qu'est-ce que votre programme doit faire ?)
2) Pour en dégager les principales tâches de programmation à accomplir et se les répartir ;
3) Travailler individuellement sur la (ou les) tâche(s) à réaliser ;
4) Mutualiser régulièrement les avancées de chacun pour effectuer des ajustements si nécessaire (ce qui est en général le cas !) ;
5) Une fois chaque tâche de programmation effectuée, réunir les différents morceaux de code de chacun pour constituer le programme final ;
6) Analyser le produit final et envisager des améliorations (sans toutefois les mettre en œuvre si manque de temps).

Pour vous simplifier la tâche, je vais vous présenter les étapes 1 et 2. Ce sont les étapes primordiales dans la réussite d'un projet, si elles ne sont pas faites correctement, il y a peu de chance que le projet aboutisse. Je vous guiderai ensuite sur l'étape 3 et vous serez chargés d'accomplir le reste des étapes.

Etape 1 : Réfléchir à la structure du programme à écrire

Réfléchissons à la structure du mode facile (nombre illimité de tentatives). Le programme devra être capable de réaliser les choses suivantes :

1. Choisir un nombre entier au hasard entre 0 et 100.
2. Récupérer les réponses du joueur.
3. Comparer la réponse du joueur au juste prix
4. Montrer au joueur où il en est dans la partie (« c'est plus », « c'est moins », « Félicitations ! »)
5. Terminer la partie quand le joueur a deviné le nombre mystère. Première analyse :

- La fonctionnalité 1 : n'est pas bien compliquée en important la bibliothèque random.
- Les fonctionnalités 2, 3 et 4 : vont être répétées un certain nombre de fois inconnu à l'avance donc il faudra utiliser une boucle tant que.
- La fonctionnalité 5 : n'est pas bien compliquée puisque la fin de la partie sera gérée par la sortie de la boucle tant que.

Voici une version algorithmique du jeu en français pour vous rendre compte à quoi cela ressemble :

Comment gérer la boucle du jeu ?
La condition dans la boucle tant que est « le joueur n'a pas deviné le nombre mystère ». Elle se traduit assez naturellement par « la réponse du joueur est différente du juste prix ». Ainsi, si on appelle rep la variable contenant la réponse du joueur et juste_prix la variable contenant le nombre entier choisi au hasard à la première ligne, alors cette condition devient : rep ≠ juste_prix.
Il reste néanmoins un problème : au premier passage dans la boucle tant que la variable juste_prix a une valeur mais la variable rep n'existe pas encore puisque le joueur n'a pas encore répondu à ce stade. Pour régler cela, il suffit d'initialiser rep à une valeur différente de juste_prix de façon à rentrer dans la boucle tant que : on peut par exemple initialiser rep à −1.
L'algorithme en pseudo-code pourrait donc être le suivant :

Analyse des instructions dans la répétitive tant que :

- Les trois fonctionnalités (2, 3 et 4) ont été transformées en des fonctions ;
- Ces trois fonctions seront donc à définir par une spécification précise et par leur algorithme en pseudo-code ;
- C'est grâce à ce « découpage » en fonctions que vous allez pouvoir vous répartir facilement les tâches de programmation et travailler de manière assez indépendante.

Etape 2 : Dégager les principales tâches de programmation à accomplir et se les répartir

Ce travail vient d'être expliqué, on peut donc découper le travail à faire en trois tâches, chacune correspondant à une des fonctions recupere_reponse, compare_reponse et montre_au_joueur
Comme vous allez travailler à trois, chacun sera en charge de l'une des tâches, c'est vous qui choisissez.
Remarque : ici, elles sont assez similaires, mais dans un projet plus important, la répartition se fait en général selon les goûts et aptitudes de chacun.
Se coordonner sur la nature des entrées et des sorties de chaque fonction
Avant de démarrer le travail individuel, il reste une chose importante à effectuer : si vous observez bien le pseudo-code, vous devez constater que les fonctions compare_reponse et montre_au_joueur ont des paramètres qui sont des valeurs renvoyées par une autre fonction. Plus précisément :

- la fonction compare_reponse prend en paramètre la variable rep qui est la valeur renvoyée par la fonction recupere_reponse ;
- la fonction montre_au_joueur prend en paramètre la variable compa qui est la valeur renvoyée par la fonction compare_reponse.

Vous devez donc vous mettre d'accord tout de suite sur la nature des valeurs renvoyées ! Sinon, il y a de fortes chances que vos fonctions soient incompatibles, par exemple : celui en charge de la fonction montre_au_joueur considère que la variable compa est de type int alors que celui en charge de la fonction compare_reponse va renvoyer une variable de type str ; l'un des deux devra recommencer son travail car les fonctions seront incompatibles.
Comme c'est votre premier (mini-)projet, je vais vous imposer ces éléments dans la suite (voir la spécification des fonctions ci-après), mais à l'avenir ce sera à vous de faire ce travail.

Etape 3 : Travailler individuellement sur la (ou les) tâche(s) à réaliser

Tâche n°1 : la fonction recupere_reponse
Spécification :

- Entrées (= paramètres) : aucune
- Sortie (= valeur(s) renvoyée(s)) : un entier r
- Rôle : demander à l'utilisateur de saisir un entier compris entre 0 et 100 et renvoyer cette valeur
- Précondition : aucune
- Postcondition : r est l'entier saisit par l'utilisateur

Tâche n°2 : la fonction compare_reponse
Spécification :

- Entrées (= paramètres) : deux entiers m et n dans cet ordre
- Sortie (= valeur(s) renvoyée(s)) : une chaine de caractères c
- Rôle : comparez les valeurs des entiers m et n
- Précondition : aucune
- Postcondition : c vaut : "=" si m = n ; vaut "+" si m < n ; vaut "-" si m > n

Tâche n°3 : la fonction montre_au_joueur
Spécification :

- Entrées (= paramètres) : une chaine de caractères c
- Sortie (= valeur(s) renvoyée(s)) : aucune
- Rôle : affiche « Félicitations ! » si c = "=" ; affiche « C'est plus » si c = "+" ; affiche « C'est moins » si c = "-".

- Précondition : la longueur de c vaut 1
- Postcondition : aucune

A vous de jouer !

Vous avez désormais toutes les cartes en mains pour terminer le mode facile.
Il vous reste également le mode difficile à concevoir par vous-même. Pour ne rien vous cacher, il suffira de modifier un peu ce qui aura été fait dans le mode facile.
N'oubliez pas de réunir ensuite les deux modes dans un même script Python, le joueur ayant la possibilité de choisir à travers un menu le mode qu'il souhaite.

Sujet 1 : niveau facile

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet01.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (5 minutes).

Sujet

Exercice 1 Fonction et conditionnelles

1. Écrire une fonction max2(a, b) qui renvoie le maximum des arguments entiers a et b sans utiliser la fonction built-in max.

Mettre entre triples guillemets, les tests effectués pour vérifier la correction de cette fonction.

```
>>> max2(10, 8)
10
```

2. Écrire une fonction max3(a, b, c) qui renvoie le maximum des arguments entiers a et b sans utiliser la fonction built-in max.

Mettre entre triples guillemets, les tests effectués pour vérifier la correction de cette fonction.

```
>>> max3(10, 8, 10)
10
```

Exercice 2 Boucle interactive

Écrire un programme qui demande d'abord à l'utilisateur un nombre entier de notes tant que le nombre saisi n'est pas strictement positif.

Ensuite le programme demande à l'utilisateur de saisir chaque note et affiche en sortie la note maximale et le nombre de fois où elle est atteinte.

Interdiction d'utiliser les listes/tableaux Python.

Voici une trace d'exécution du programme :

```
Nombre de notes ? -2

Saisir un nombre > 0, nombre de notes ? -1

Saisir un nombre > 0, nombre de notes ? 3

Nouvelle note ? 10

Nouvelle note ? 8

Nouvelle note ? 10
Note maximale : 10.0 atteinte 2 fois
```

Sujet 2 : niveau facile

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet02.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme), de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 3 points pour la prestation orale (différencié par binôme).

Exercice 1 Fonction et conditionnelles

1. Écrire une fonction bac(moyenne) qui prend en argument une moyenne de type float entre 0 et 20, vérifie si la moyenne est entre 0 et 20 avec une instruction assert et renvoie :
   - True si moyenne >= 10
   - False si moyenne < 10
2. Écrire une fonction mention(moyenne) qui prend en argument une moyenne de type float entre 0 et 20 et qui renvoie une chaîne de caractère :
   - "recalé" si $0 \leqslant$ note < 8
   - "second groupe" si $8 \leqslant$ note < 10
   - "reçu" si $10 \leqslant$ note < 12
   - "assez bien" si $12 \leqslant$ note < 14
   - "bien" si $14 \leqslant$ note < 16
   - "très bien" si $16 \leqslant$ note $\leqslant 20$
   - "valeur incohérente" sinon

On donne quelques tests unitaires qui doivent être vérifiés. Compléter ces tests en essayant de couvrir tous les cas possibles. Mettre entre triples guillemets, les tests effectués.

```
>>> assert mention(9) == "second groupe"
>>> assert mention(-1) == "valeur incohérente"
```

Exercice 2 Boucle interactive

Écrire un programme qui demande d'abord à l'utilisateur un nombre entier de notes tant que le nombre saisi n'est pas strictement positif.
Ensuite le programme demande à l'utilisateur de saisir chaque note et affiche en sortie la moyenne des notes saisies, arrondie au centième. On utilisera la fonction round, pour afficher sa documentation, saisir help(round) dans une console interactive.
Attention! Interdiction d'utiliser les listes/tableaux Python.
Voici une trace d'exécution du programme :

```
Nombre de notes ? -2

Saisir un nombre > 0, nombre de notes ? 3

Nouvelle note ? 10

Nouvelle note ? 8

Nouvelle note ? 5
Moyenne : 7.67
```

Sujet 3 : niveau facile
Ouverture et lecture d'un fichier avec Python

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet03.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme), de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 3 points pour la prestation orale (différencié par binôme).

Ce mini-projet est guidé. À vous de suivre les instructions :)

Pour les exemples, on va utiliser le fichier `exemple.txt` (voir lien à la fin du sujet).

Q1 : Ouvrez le fichier `exemple.txt` avec un éditeur de texte pour en observer le contenu.

Pour ouvrir ce fichier avec Python on utilise la fonction `open()` comme ci-dessous :

```
f = open('exemple.txt', 'r', encoding='utf8')
```

Analyse : On a passé 3 arguments à cette fonction :

- le nom du fichier `'exemple.txt'` ;
- le mode d'ouverture : `'r'` (première lettre de *read* en anglais, c'est-à-dire "lecture") qui indique que l'on ouvre le fichier juste pour lire son contenu (et pas écrire dedans par exemple) ;
- l'encodage utilisé : `encoding='utf8'`. Nous renviendrons là-dessus plus tard dans l'année, ce n'est pas important pour le moment.

Cette fonction `open()` renvoie un "objet fichier" que l'on va pouvoir manipuler avec Python. On peut notamment lire *tout* le contenu du fichier et stocker cela dans une variable appelée ici `contenu` :

```
f = open('exemple.txt', 'r', encoding='utf8')  # ouverture
contenu = f.read()  # lecture
f.close()  # fermeture du flux de lecture

contenu # pour voir la valeur de la variable contenu
```

Attention : Il ne faut pas oublier de fermer le flux de lecture pour que le fichier reste accessible pour les autres applications.

Q2 : En observant la sortie ci-dessus, indiquez le type de la variable `contenu` en justifiant. Quelle instruction Python permet de vérifier votre réponse ? Faites-le.

Q3 : Par quoi ont été convertis les caractères invisibles de fin de ligne du fichier texte ?

Réponse : (à compléter)

Notez que la fonction `print()` permet d'afficher le contenu comme un éditeur de texte :

Reste du projet
https://info-mounier.fr/premiere_nsi/projets/data/MP_AnalyseFichierTexte/MP_AnalyseFichierTexte.zip

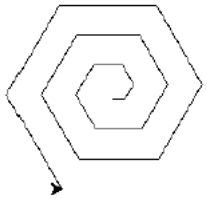
Sujet 4 : niveau moyen

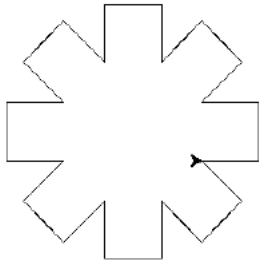
Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet04.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme), de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 3 points pour la prestation orale (différencié par binôme).

Exercice 1 Boucles imbriquées et tortue

Écrire avec le module turtle deux programmes permettant de réaliser les figures ci-dessous en un minimum de lignes (from turtle import *) compris.





Exercice 2

Le Queulorior a ceci de particulier que de la peinture s'écoule au bout de sa queue. Ainsi, lorsqu'il se déplace, il laisse une trace.

Sacha sait que vous avez un Queulorior et pour communiquer avec vous, il vous envoie des instructions à donner au Queulorior. Ces instructions forment une séquence assez longue composée de 4 lettres : `N`, `S`, `O` et `E`. La lettre `N` indique au Queulorior de faire un pas vers le nord, la lettre `S` lui indique de faire un pas vers le sud, la lettre `E` un pas vers l'est et la lettre `O` un pas vers l'ouest.

Par exemple, donnez ces instructions au Queulorior :

```
NNEESOOEES
```

Il va faire deux pas vers le nord, deux pas vers l'est, un pas vers le sud, deux pas vers l'ouest, deux pas vers l'est et un pas vers le sud, ce qui dessinera un A approximatif :

Vous devez lire le message de Sacha, mais souhaitez économiser un peu votre Queulorior. Vous allez donc devoir déchiffrer le message vous-même.

Pour valider le défi, indiquez où est le rendez-vous avec Sacha.

Attention à ne pas prendre les B pour des A dans le message

.

Sujet 6 : niveau difficile Jouer avec les anagrammes

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exer- cices en le nommant selon le format NOM1_NOM2_Sujet06.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme), de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 3 points pour la prestation orale (différencié par binôme).

Exercice 1

Rédigez une function nommée is_anagram, prenant deux chaînes, et renvoyant un booléen.

Cette fonction doit renvoyer True si un mot peut être obtenu par une permutations des lettres de l'autre.

Des espaces, apostrophes, tirets superflus sont autorisés, par exemple : funeral est une anagramme de real fun.
Les majuscules sont ignorées, Marion est une anagramme de Romain.
Les accents sont ignorés, Noé et Néo sont des anagrammes de One.

Conseils

N'utilisez pas la bibliothèque unidecode : elle n'est pas installée sur le serveur de correction.

N'utilisez pas non plus un grand dictionnaire ou une grande série de « .replace » : ce n'est pas élégant et vous allez oublier des lettres comme Ÿ, ṅ, j̄…

Votre but est simplement d'enlever les diactritiques, la bonne méthode pour ça est de passer par la forme normale décomposée de la chaîne de caractères, puis de retirer les caractères de composition.

Pour passer à la forme normale décomposée utilisez la fonction normalize, et pour savoir si un caractère est un caractère de combinaison utilisez la fonction combining.
Voici une fonction qui pourra être utile :

```
def remove_accent(mot):
    """Renvoie un mot avec les mêmes caractères mais sans accents"""
    accents = {"à": "a","ä": "a","â": "a",
               "é": "e","è": "e","ê": "e",
               "ë": "e","î": "i","ï": "i",
               "ô": "o","ö": "o","û": "u",
               "ü": "u","ù": "u","ç": "c",}
    mot2 = ""

    for c in mot:
        if c in accents:
            mot = mot + accents[c] else:
            mot = mot + c return mot2

assert remove_accent('préconçue') == 'preconcue'
assert remove_accent('inouïe') == 'inouie'
```

Sujet 7 : niveau moyen La carte manquante

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet07.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme), de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 3 points pour la prestation orale (différencié par binôme).

Exercice 1 Hackinscience

Rédigez une fonction nommée `missing_card` acceptant un jeu de carte dont une carte manque, et renvoyant le nom de la carte manquante.

Le jeu de carte sera donné à votre fonction sous forme d'une chaîne de noms de carte séparés par des espaces.

Un nom de carte contient la couleur et la valeur de la carte. La couleur est dans l'ensemble `{"S", "H", "D", "C"}` et sa valeur dans `{"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"}`, soit un total de 52 cartes.

Votre fonction recevra toujours 51 cartes, elle doit trouver la carte manquante.

Exemple

```
print(
  missing_card(
    "S2 S3 S4 S5 S6 S7 S8 S9 S10 SJ SQ SK SA "
    "H2 H3 H4 H5 H6 H7 H8 H9 H10 HJ HQ HK HA "
    "D2 D3 D4 D5 D6 D7 D8 D9 D10 DJ DQ DK DA "
    "C2 C3 C4 C5 C6 C7 C8 C9 C10 CJ CQ CK"
  )
)
```

doit afficher `CA` (attention, votre fonction doit renvoyer `CA`, dans mon exemple j'ai utilise `print` pour l'afficher).

Sujet 8 : niveau facile : Les aventures d'Hercule

Consignes

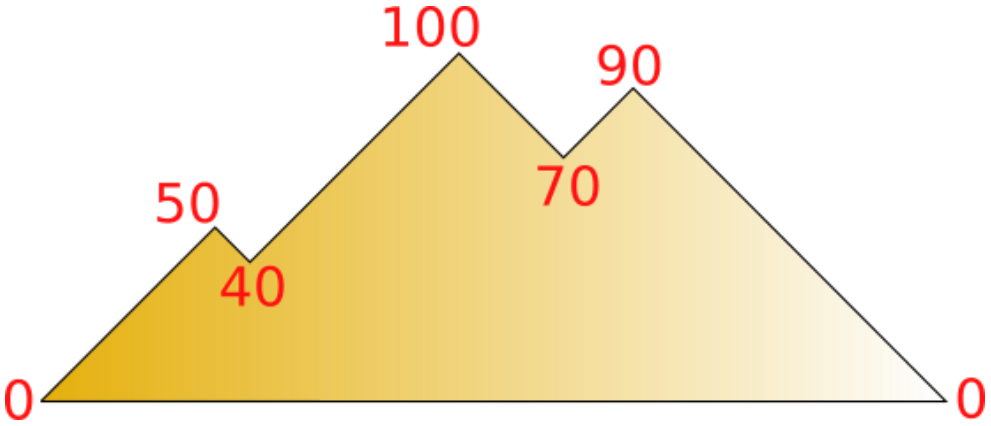
- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet08.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Exercice 1

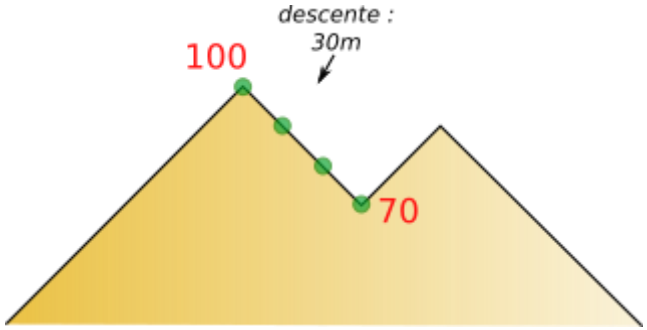
Histoire

Pour son 4e travail, Eurysthée demanda à Hercule de capturer vivant le sanglier d'Érymanthe. Gigantesque, celui-ci dévastait avec rage le nord-ouest de l'Arcadie.

Après avoir débusqué le sanglier, Hercule le poursuivit dans les montagnes en lui jetant des pierres. Le profil montagneux était assez accidenté et ressemblait à ceci :

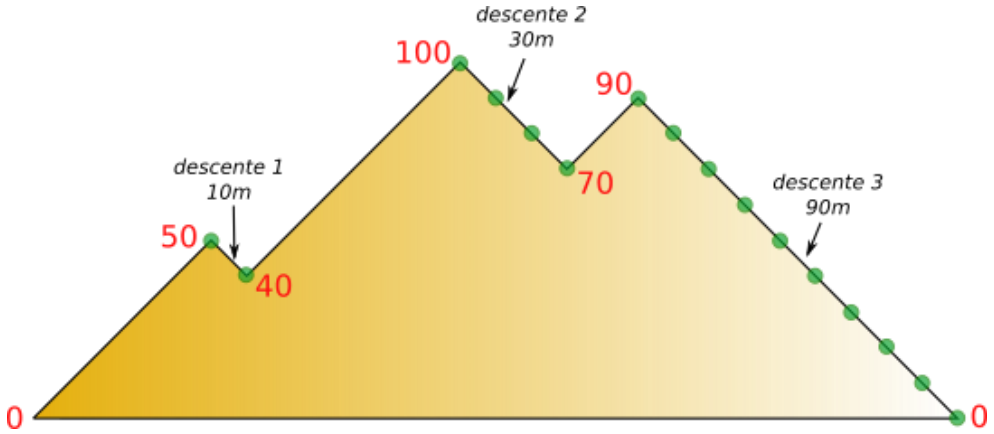


Hercule, pour économiser ses forces, ne jetait des pierres que dans les descentes. Précisément, il jetait une pierre tous les 10 mètres (changement d'altitude). Ainsi, dans une descente de 30 mètres, il jetait 4 pierres.



On peut produire un relevé du profil des montagnes, en donnant les altitudes de chaque sommet et chaque col. Dans l'exemple qui précède, le relevé donnerait : `0, 50, 40, 100, 70, 90, 0`

À partir de ce relevé uniquement, on peut voir qu'il y a 3 descentes, de 10, 30 et 90 mètres :



Hercule jettera donc 16 pierres sur le sanglier.

Défi

Un relevé du profil réel vous est donné en entrée. Vous devez indiquer à Hercule combien de pierre il aura à jeter sur le sanglier.

Exercice 2

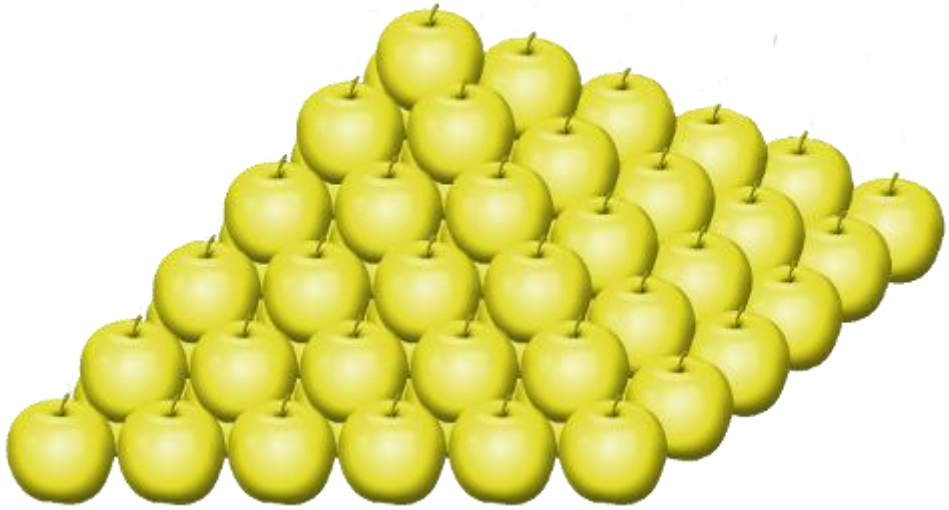
Histoire

Les Hespérides, filles d'Atlas, habitaient un merveilleux jardin dont les pommiers donnaient des pommes en or. Pour son 11e travail, Eurysthée demanda à Hercule de ramener ces pommes.

Une fois atteint le jardin merveilleux, l'oracle Nérée apprit à Hercule qu'il pourrait repartir avec une partie des pommes... à condition qu'il montre ses facultés en calcul mental. Nérée lui tint ce propos :

J'ai empilé les pommes d'or pour toi, sous la forme d'une pyramide. L'étage le plus haut ne contient qu'une pomme. L'étage juste en dessous forme un carré 2x2 (contenant 4 pommes), l'étage juste en dessous forme un carré 3x3 (contenant 9 pommes). La pyramide que tu vois contient 50 étages. L'étage de base contient donc 2 500 pommes... Je suis d'accord pour te laisser partir avec les pommes contenues dans certains étages. Précisément, si un étage contient un nombre de pommes multiple de 3, tu peux l'emporter. Si tu m'annonces combien de pommes tu emporteras au total, je te laisserai partir avec les pommes...

Pour relever ce défi, vous devez aider Hercule en lui indiquant le nombre de pommes qu'il pourra emporter. Par exemple, si la pyramide n'avait compté que 6 étages comme indiqué sur la figure suivante, chaque étage aurait été composé de : 1, 4, 9, 16, 25 et 36 pommes. Hercule aurait pu emporter les 9 pommes de l'étage 3 (car 9 est un multiple de 3) et les 36 pommes de l'étage 6 (car 36 est un multiple de 3). Au total il aurait donc emporté 45 pommes. Mais combien peut-il en emporter pour une pyramide de 50 étages ?



Sujet 9 : niveau moyen Dessiner une grille

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet09.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Exercice 1 Hackinscience

Créer la fonction `draw_n_squares(n)` qui renvoit un texte « dessinant » un quadrillage comme suit:

```
>>> from solution import draw_n_squares

>>> print(draw_n_squares(1))
+---+
|   |
+---+

>>> print(draw_n_squares(3))
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
```

```
>>> print(draw_n_squares(5))
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
```

`n` est le nombre de carrés sur la diagonale.

Sujet 10 : niveau difficile : Les nombres pernicieux

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet10.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Numéro pernicieux

En théorie des nombres , un **nombre pernicieux** est un entier positif tel que le poids de Hamming de sa représentation binaire est premier .

Exemples

Le premier nombre pernicieux est 3, puisque 3 = 11_2 et 1 + 1 = 2, qui est un nombre premier. Le nombre pernicieux suivant est 5, puisque 5 = 101_2 , suivi de 6, 7 et 9 (séquence A052294 dans l' OEIS).

Exercice

Rédigez un programme ne prenant aucun paramètres.

Ce programme doit afficher les nombres pernicieux (en) dans l'intervalle `range(222281, 222381)`

Cet exercice peut être compliqué si vous ne comprennez pas parfaitement ce qu'est un nombre pernicieux, et puisqu'aucun exercice n'est obligatoire pour passer à la suite, sentez-vous libre de le sauter.

Affichez un et un seul nombre par ligne, rien d'autre.

Si le but de l'exercice était d'afficher les nombres pernicieux de l'intervalle `range(100, 110)`, on verrait :

```
$ ./solution.py
100
103
104
107
109
```

Parce que :

- 100 vaut 1100100 en base 2, qui est composé d'un nombre premier de uns (3).
- 103 vaut 1100111 en base 2, qui est composé d'un nombre premier de uns (5).
- 104 vaut 1101000 en base 2, qui est composé d'un nombre premier de uns (3).
- 107 vaut 1101011 en base 2, qui est composé d'un nombre premier de uns (5).
- 109 vaut 1101101 en base 2, qui est composé d'un nombre premier de uns (5).

Sujet 11 : niveau moyen

La plus longue séquence de Collatz La suite de Fibonacci

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet11.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Exercice 1

The collatz sequence goes like this:

- Start by any number greater than zero.
- If your number is even, divide it by two.
- If your number os odd, multiply by three and add one.

For example, starting by `10` we have:

```
10 → 5 → 16 → 8 → 4 → 2 → 1
```

Write a function, named `collatz_length`, given a number the function return the length of the sequence before reaching 1, for example:

```
>>> collatz_length(10)
6
```

Because it takes 6 steps (6 → in the previous example) to reach 1.

Exercice 2

Provide a function that returns the `n` first numbers of the fibonacci sequence.

Name your function `fibonacci` and take a single parameter, say `n`.

Your function should return an iterable of `n` values from the sequence.

Examples

```
>>> fibonacci(2)
[1, 1]
>>> fibonacci(5)
[1, 1, 2, 3, 5]
```

Sujet 12 : niveau moyen - Du romain au décimal

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet12.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Exercice 1

Rédigez une fonction nommée `from_roman_numeral` renvoyant la valeur décimal de nombre romain reçu en paramètre.

Exemple

```
>>> print(from_roman_numeral("V"))
5
>>> print(from_roman_numeral("XX"))
20
>>> print(from_roman_numeral("DCCC"))
800
>>> print(from_roman_numeral("MMMM"))
4000
```

Sujet 13 : niveau moyen

Mélange à queue d'aronde Distance

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet13.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Exercice 1

Simulez un « mélange à queue d'aronde » sur un paquet de cartes.

Le mélange à queue d'aronde consiste à couper le jeu en deux moitiées égales, puis de les intercaler parfaitement : une du tas de gauche, une du tas de droite, une du tas de gauche…

Mélanger `[1, 2, 3, 4, 5, 6]` donne donc `[1, 4, 2, 5, 3, 6]`.

Votre fonction ne recevra que des jeux de cartes contenant un nombre pair de cartes.

Votre fonction devra s'appeler `perfect_shuffle(deck)`.

Saviez vous que …

…mélanger 10 fois un paquet de 1024 cartes revient à ne rien faire ?

C'est probablement un bon moyen de tester votre implémentation.

Exercice 2

Rédigez une fonction trouvant la distance entre les deux nombres les plus éloignés d'une liste donnée.

Créez une fonction nommée `dist`, acceptant un paramètre : une liste de nombres (entiers ou flotants), et renvoyant la plus grande distance entre deux nombres de cette liste, typiquement entre le plus grand et le plus petit.

```
>>> dist([1, 2, 3])
2
>>> dist([1, 2, 3, 2.5])
2
>>> dist([1, 2, 3, 2.5, 3.5])
2.5
>>> dist([1, 2, 3, 2.5, 3.5, 120])
119
>>> dist([1, 2, 3, 2.5, 3.5, 120, -1000])
1120
```

Sujet 14 : niveau moyen

Consignes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet14.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme),de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 1 point pour la réalisation des objectifs fixés, 3 points pour la prestation orale (différencié par binôme).

Pour connaitre l'essentiel sur le language morse :
https://www.techno-science.net/definition/3815.html

Exemple

```
-*-* --- -** *       /      -- --- *-* *** *
C    O   D   E   (espace)   M  O   R   S   E
```

Utilisation d'un dictionnaire

On représente le code morse à l'aide d'un dictionnaire, on ne s'intéresse qu'aux lettres en majuscules non accentuées.

Pour l'espace on utilise le slash (par exemple). Vous pourrez recopier dans votre code ce dictionnaire.

Script Python

```
morse = {' ': '/', 'E': ' *', 'I': ' * *', 'S': ' * * *', 'H': ' * * * *', 'V': ' * * * –', 'U': ' * * –',
'F': ' * * – *', 'A': ' * –', 'R': ' * – *', 'L': ' * – * *', 'W': ' * – –', 'P': ' * – – *', 'J': ' * –
– –', 'T': ' –', 'N': ' – *', 'D': ' – * *', 'B': ' – * * *', 'X': ' – * * –', 'K': ' – * –', 'C': ' –
* – *', 'Y': ' – * – –', 'M': ' – –', 'G': ' – – *', 'Z': ' – – * *', 'Q': ' – – * –', 'O': ' – –
–'}
```

Travail à faire :

1. Ecrire un script qui permet de chiffrer un message en morse

2. Ecrire un script qui permet de déchiffrer un message envoyé en morse.

Exemples d'exécution

Vous pouvez avoir des noms de fonctions différents. Ceci n'est qu'un exemple.

Console Python

```
>>> message1 = 'VIVE LA NSI'
>>> code_mots(message1, morse)
'***- ** ***- */*-** *-/-* *** **'

>>> message2 =
'-*** *-* *- ***- ---/ *--- * **- -* */ *--* *- -** *- *-- *- -*/ *-*** *-/ -
* *** **/ * *** -/ *- ***- * -*-*/ - ---**'
>>> decode_mots(message2, morse)
'BRAVO JEUNE PADAWAN LA NSI EST AVEC TOI'
```

Astuce : fonction split

Sujet 15 : niveau facile

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet15.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme),de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 1 point pour la réalisation des objectifs fixés, 3 points pour la prestation orale (différencié par binôme).

Sujet

Un point est parfaitement défini dans un repère par ses coordonnées. On peut étendre cette notion en python en utilisant des tuples. Par exemple un point $A(3;2)$, peut se définir en python par : >>> A=(3,2)

On peut alors définir des fonctions qui utiliseront le cours de mathématiques de la classe de seconde. Par exemple, on sait que le milieu M d'un segment [AB] a pour coordonnées $\left(\frac{x_A+x_B}{2};\frac{y_A+y_B}{2}\right)$ On peut alors facilement définir une fonction qui déterminera les coordonnées du milieu de deux points passés en paramètres :

```
def milieu(point1,point2):
  return ((point1[0]+point2[0])/2,(point1[1]+point2[1])/2)
```

On a alors :

```
>>> A=(1,1)
>>> B=(-1,-1)
>>> milieu(A,B)
(0.0, 0.0)
```

Exercice 1

Définir une fonction vecteur correspondant à sa docstring :

```
def vecteur(p1,p2):
    ''' retourne les coordonnées du vecteur d'origine p1 et d'extrémité p2
    return : tuple de longueur 2 (x,y)

    >>> vecteur((3,2),(1,-5)
    (-2,-7)
    >>> A=(3,2)
    >>> B=(1,1)
    >>> vect(A,B)
    (-2,-1)
    >>> vect(B,A)
    (2,1)
    '''
```

Exercice 2

On considère des points repérés par leurs coordonnées entières dans un repère. Ajouter à votre programme des fonctions permettant de :

- déterminer si 4 points forment un parallélogramme,
- déterminer si deux vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CD}$ sont colinéaires.
- déterminer si 3 points sont alignés,
- déterminer une valeur approchée de la distance entre 2 points, (pour un repère orthonormé)
- déterminer l'équation d'une droite passant par 2 points, etc.....

Le travail se fera par groupe de deux au maximum, chacun s'occupant de définir une ou plusieurs fonctions.

Sujet 16 : niveau facile

Logiciel d'apprentissage des tables de multiplication

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet16.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).
- Barème sur 10 points : 1 point pour l'activité de recherche en classe (différencié par binôme),de 1 (facile), 2 (moyen), à 3 (difficile) points de réalisation d'objectif, 3 points pour la qualité et la correction du code, 1 point pour la réalisation des objectifs fixés, 3 points pour la prestation orale (différencié par binôme).

Sujet : Le but de ce projet est de créer un logiciel éducatif permettant de s'entraîner sur les tables de multiplication.

Cahier des charges

Votre programme demande d'abord à l'utilisateur quelle table de multiplication il veut réviser. Une fois qu'il a répondu, le programme lui pose 10 questions de la forme : "Combien font 3 x 5 ?" correspondant à la table choisie. Après chaque question, le programme indique si la réponse est fausse (et ne dit rien si elle est juste). À la fin, le programme affiche le nombre de bonnes réponses sur 10 et félicite l'utilisateur s'il a fait un sans-faute. S'il n'a pas fait un sans-faute, le programme fait essayer l'utilisateur à nouveau.

Exemple d'exécution

Voici un exemple d'exécution de votre programme:

```
Quelle table de multiplication voulez-vous réviser ?
5
Combien font 5 x 1 ? 5
Combien font 5 x 2 ? 10
Combien font 5 x 3 ? 15
Combien font 5 x 4 ? 22
Erreur : la réponse était 20
Combien font 5 x 5 ? 25
Combien font 5 x 6 ? 30
Combien font 5 x 7 ? 35
Combien font 5 x 8 ? 39
Erreur : la réponse était 40
Combien font 5 x 9 ? 45
Combien font 5 x 10 ? 50

Votre note est de 8/10

Essayons à nouveau !
Quelle table de multiplication voulez-vous réviser ?
5       et puis ça continue encore jusqu'à un 10/10
```

Pour aller plus loin (palier 4)

Une fois que vous avez atteint le palier 3 des fonctionnalités, voici quelques idées pour aller plus loin et atteindre le palier 4 :

- Demander les dix questions dans un ordre aléatoire (et non pas toujours dans le même ordre), sans répétition.
- Proposer à l'utilisateur soit de choisir une table soit que le programme en choisisse une au hasard.
- Proposer à l'utilisateur que les dix questions puissent provenir de tables différentes (choisies au hasard).

Sujet 17 : niveau facile Analyseur de mots de passe

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet17.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet : Le but de ce projet est de créer un assistant qui analyse la robustesse d'un mot de passe.

Cahier des charges

Votre programme demande d'abord à l'utilisateur de saisir un mot de passe. Si le mot de passe ne suit pas les règles, le programme indique pourquoi à l'utilisateur et l'invite à en saisir un nouveau. Un mot de passe ne suit pas les règles dans les cas suivants :

- il est trop court (4 caractères ou moins) ;
- il est trop long (plus de 20 caractères) ;
- il ne contient pas de majuscule ;
- il ne contient pas de minuscule ;
- il ne contient pas de chiffres.

Si le mot de passe suit les règles, le programme affiche un message indiquant que le mot de passe est valide et demande à l'utilisateur de l'entrer à nouveau. Si les deux mots de passe sont identiques, le programme affiche un message de confirmation. Sinon, le programme recommence depuis le début.

Exemple d'exécution

Voici un exemple d'exécution de votre programme :

```
Saisissez votre mot de passe : Top NSI
Ce mot de passe n'est pas sécurisé : il ne contient pas de
chiffres.
Saisissez votre mot de passe : T0p NS1
Ce mot de passe est sécurisé.
Saisissez à nouveau votre mot de passe : Tob NS1
Erreur : les mots de passe ne sont pas identiques. Merci de
recommencer.

Saisissez votre mot de passe : t0p ns1
Ce mot de passe n'est pas sécurisé : il ne contient pas de
majuscule.
Saisissez votre mot de passe : T0p NS1
Ce mot de passe est sécurisé.
Saisissez à nouveau votre mot de passe : T0p NS1
Le mot de passe est correct !
```

Pour aller plus loin (palier 4)

Une fois que vous avez atteint le palier 3 des fonctionnalités, voici quelques idées pour aller plus loin et atteindre le palier 4 :

- Demander d'insérer un ou plusieurs caractères spéciaux.
- Demander plusieurs mots de passe à l'utilisateur en associant à chaque fois un nom de site (par exemple, "Facebook", "Twitter", "Instagram", etc.). Le programme interdit alors à l'utilisateur d'entrer deux fois le même mot de passe pour des services différents. Il lui propose aussi, au lieu de saisir un mot de passe, d'afficher un mot de passe déjà enregistré pour un service donné.

Sujet 18 : niveau facile Détection de palindromes

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet18.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet : Un palindrome est un mot ou une phrase qui se lit de la même façon dans les deux sens. Par exemple, « radar » est un mot palindrome, et « Ésope reste ici et se repose » est une phrase palindrome.

Le but de ce projet est de créer un programme qui détecte si un mot ou une phrase est un palindrome.

Cahier des charges

Votre programme demandera à l'utilisateur de saisir un mot ou une phrase. Il indiquera ensuite si le mot ou la phrase est un palindrome ou non selon différents niveaux de sensibilité :

- **palindrome strict** - la chaîne de caractères est l'exact miroir d'elle-même (sans tenir compte des majuscules et des minuscules) :
  - `Été` est un palindrome strict
  - `Radar` est un palindrome strict
  - `Esope reste ici et se repose` n'est pas un palindrome strict
- **palindrome sans espaces** - si on enlève les espaces, la chaîne est un palindrome strict :
  - `Karine alla en Irak` est un palindrome sans espaces
  - `Esope reste ici et se repose` est un palindrome sans espaces
  - `Ésope reste ici et se repose` n'est pas un palindrome sans espaces (à cause de l'accent sur le « E »)
- **palindrome flexibles** - si on enlève les espaces, les cédilles, les accents et la ponctuation, la chaîne est un palindrome strict :
  - `Ésope reste ici et se repose.` est un palindrome flexible
  - `La mère Gide digère mal.` est un palindrome flexible
  - `Eh ! ça va la vache ?` est un palindrome flexible
  - `Ésope reste ici et se repose` est un palindrome flexible

Exemples d'exécution

Voici des exemples d'exécution de votre programme (le texte affiché par le programme est en bleu, tandis que le texte entré par l'utilisateur est en noir).

Pas un palindrome :

```
Entrez un mot ou une phrase : Bonjour
Pas un palindrome !
```

Palindrome strict :

```
Entrez un mot ou une phrase : Kayak
Palindrome strict !
```

Palindrome sans espaces :

```
Entrez un mot ou une phrase : Karine alla en Irak
Palindrome sans espaces !
```

Palindrome flexible :

```
Entrez un mot ou une phrase : Eh ! ça va la vache ?
Palindrome flexible !
```

Pour aller plus loin (palier 4)

Une fois que vous avez atteint le palier 3 des fonctionnalités, voici quelques idées pour aller plus loin et atteindre le palier 4 :

- Indiquer quand un mot est presque un palindrome (indiquer s'il y a une lettre de différence, deux lettres de différence, etc.).
- Ajouter une fonctionnalité pour détecter les palindromes dans le dictionnaire (en s'aidant du fichier suivant : dictionnaire). Votre programme génère alors une liste de palindromes stricts, une liste de palindromes sans espaces et une liste de palindromes flexibles.

Sujet 19 : niveau facile Le code de César

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet19.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet :

Le code de César est une manière de chiffrer un message de manière simple : On choisit un nombre (appelé clé de chiffrement) et on décale toutes les lettres de notre message du nombre choisi.

Par exemple, si on choisit comme clé le nombre 4 alors la lettre A devient E, le B devient F, ... et le Z devient D. Ainsi, le mot ZEBRE est chiffré en DIFVI.

On remarque que pour décoder un message chiffré avec le code de César, il suffit de choisir la clé inverse (ici -2) et de réappliquer le décalage.

Le but de ce projet est de créer un programme qui permet de chiffrer et de déchiffrer des messages avec le code de César.

Cahier des charges

Votre programme demande à l'utilisateur de saisir une phrase à chiffrer ou déchiffrer, ainsi que la clé (un nombre entier, positif ou négatif). Il affiche ensuite le message chiffré ou déchiffré.

Votre programme doit pouvoir gérer les majuscules et les minuscules. Par exemple, si l'utilisateur saisit `Bonjour` et la clé 2, votre programme doit afficher `Dpncqpr`.

De plus, votre programme doit pouvoir gérer les caractères spéciaux (espaces, ponctuation, ...). Par exemple, si l'utilisateur saisit `Bonjour, comment ca va ?` et la clé 2, votre programme doit afficher `Dpncqpr, eqoovg ec xc ?`.

Exemples d'exécution

Voici des exemples d'exécution de votre programme (le texte affiché par le programme est en bleu, tandis que le texte entré par l'utilisateur est en noir).

Exemple 1 :

```
Entrez un message : ZEBRE
Entrez la clé : 4
DIFVI
```

Exemple 2 :

```
Entrez un message : Dpncqpr
Entrez la clé : -2
Bonjour
```

Exemple 3 :

```
Entrez un message : Bonjour, comment ca va ?
Entrez la clé : 15
Qdcydjg, rdbbtci rp kp ?
```

Pour aller plus loin (palier 4)

Une fois que vous avez atteint le palier 3 des fonctionnalités, voici quelques idées pour aller plus loin et atteindre le palier 4 :

- Ajouter la gestion des caractères accentués (é, è, à, ç, ...). Avant de chiffrer un tel caractère, il faut lui enlever son accent.
- Chiffrement polyalphabétique : le chiffre de Vigenère. Cette méthode de chiffrement est plus complexe que le chiffre de César. Elle consiste à utiliser un mot ou une phrase au lieu d'un nombre pour chiffrer le message. On associe à chaque lettre de la clé un décalage (A=1, B=2, C=3, ...). On chiffre ensuite la première lettre du message avec la première lettre de la clé, la deuxième lettre du message avec la deuxième lettre de la clé, etc. Si la clé est plus courte que le message, on recommence à la première lettre de la clé. Par exemple, si le message est `Bonjour` et la clé `ABC`, on chiffre `B` avec `A`, `o` avec `B`, `n` avec `C`, `j` avec `A`, `o` avec `B`, `u` avec `C`, `r` avec `A`. On obtient donc le message chiffré `Copkps`. Pensez à permettre le déchiffrement avec la même clé.

Sujet 20 : niveau facile Calculatrice virtuelle

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet20.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet :

Le but de ce projet est de créer une calculatrice qui permet de faire des opérations simples sur les entiers et les nombres à virgule. Elle permet aussi de refaire une opération en partant du résultat de la dernière opération.

Cahier des charges

Votre programme demande à l'utilisateur de saisir un nombre de départ. Il affiche ensuite un menu proposant à l'utilisateur les opérations suivantes (en remplaçant x par le nombre saisi) :

- **x + y :** addition
- **x - y :** soustraction
- **x × y :** multiplication
- **x ÷ y :** division
- **√x :** racine carrée

Dans le cas d'une opération qui nécessite un deuxième nombre, l'utilisateur doit saisir ce nombre. Votre programme affiche ensuite le résultat de l'opération et propose à l'utilisateur de refaire une opération en partant du résultat de la dernière opération.

Votre programme continue de proposer à l'utilisateur de refaire une opération jusqu'à ce qu'il choisisse de quitter. Pour lui permettre de quitter, ajoutez une option **Quitter** dans le menu de choix de l'opération.

Attention : votre programme doit pouvoir gérer les entiers et les nombres à virgule. Une façon simple de savoir si un nombre saisi par l'utilisateur est un nombres à virgule est de regarder s'il y a un point (`.`) dans la chaîne de caractères. Si c'est le cas, vous pouvez convertir la chaîne de caractères en nombre à virgule avec la fonction `float()`.

Exemple d'exécution

Voici un exemple d'exécution de votre programme (le texte affiché par le programme est en bleu, tandis que le texte entré par l'utilisateur est en noir).

```
Entrez un nombre : 42
Choisissez une opération :
1) 42 + y
2) 42 - y
3) 42 × y
4) 42 ÷ y
5) √42
0) Quitter
Choix : 1
Valeur de y : 7
Résultat : 49
Choisissez une opération :
1) 49 + y
```

```
2) 49 - y
3) 49 × y
4) 49 ÷ y
5) √49
0) Quitter
Choix : 5
Résultat : 7.0
Choisissez une opération :
1) 7.0 + y
2) 7.0 - y
3) 7.0 × y
4) 7.0 ÷ y
5) √7.0
0) Quitter
Choix : 3
Valeur de y : 1.5
Résultat : 10.5
Choisissez une opération :
1) 10.5 + y
2) 10.5 - y
3) 10.5 × y
4) 10.5 ÷ y
5) √10.5
0) Quitter
Choix : 0
```

Pour aller plus loin (palier 4)

Une fois que vous avez atteint le palier 3 des fonctionnalités, voici quelques idées pour aller plus loin et atteindre le palier 4 :

- Ajouter plus d'opérations, par exemple :
  - **x div y :** division entière (quotient de la division euclidienne)
  - **x mod y :** modulo (reste de la division euclidienne)
  - **x ^ y :** exponentiation (puissance)
  - **inv(x) :** inverse (1/x)
- Proposer à l'utilisateur de repartir à 0 (en saisissant un nouveau nombre de départ).
- Créer un historique des résultats intermédiaires et permettre à l'utilisateur de choisir un résultat intermédiaire pour le nombre y grâce à un menu.

Sujet 21 : projet long Le jeu de morpion

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet21.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet :

Principe de l'algorithme

La plupart des jeux de "plateau" à deux joueurs suivent le même principe :

- Un joueur joue
- S'il a gagné, le jeu s'arrête
- Si ce n'est pas le cas, on reprend au premier point avec l'autre joueur

Ce processus se répète jusqu'à ce qu'un des joueurs remporte la partie, ou que la partie s'achève sur une égalité.

Comme on répète tour à tour le même schéma avec chaque joueur, il est intéressant de programmer ce principe par une boucle. Et comme on ne sait pas à l'avance en combien de tours la partie se termine, on utilise une boucle Tant que.

Dans le cas d'un jeu numérique, il faut également afficher l'état du plateau de jeu à chaque tour. Cet état peut être mémorisé dans un tableau à deux dimensions (un tableau de tableaux).

Dans le cas du jeu du Morpion, on peut mémoriser l'état initial du plateau du jeu dans le tableau suivant.

```python
jeu = [['.', '.', '.'],
       ['.', '.', '.'],
       ['.', '.', '.']]
```

Python

Analyse plus fine et décomposition du problème

Dans un projet, la première phase consiste toujours à **décomposer le problème en différentes tâches simples**.

Pour cela, il est nécessaire de réfléchir de manière plus fine au problème en détaillant davantage le principe de l'algorithme. C'est cette analyse qui va permettre d'identifier les différentes **tâches** attendues dans notre algorithme/programme.

Principe de l'algorithme

Voici le principe de notre algorithme :

```
On affiche le plateau
On définit qui joue en premier
Tant que la partie n'est pas gagnée :
    Le joueur joue
    On affiche le plateau après son coup
    On vérifie s'il a gagné
        Si oui, la partie est gagnée et le jeu s'arrête ;
        Sinon, c'est au tour de l'autre joueur.
```

Décomposition en tâches

On peut désormais identifier les tâches principales à accomplir pour programmer notre jeu. Chacune de ces tâches sera programmée par une fonction. Voici les fonctions principales nécessaires :

- `afficher_plateau()` : fonction qui affiche l'état du plateau de jeu à l'écran (dans la console pour nous ici) ;
- `jouer()` : fonction qui demande à l'utilisateur la case qu'il veut jouer et qui actualise le tableau `jeu` ;
- `verifier_victoire()` : fonction qui vérifie si la partie est gagnée ;
- `changer_joueur()` : fonction qui change le joueur.

On utilisera les trois variables suivantes :

- `jeu` est le tableau mémorisant l'état du plateau de jeu (voir plus haut) ;
- `joueur` est une chaîne de caractères prenant les valeurs `'X'` ou `'O'` et qui désigne le joueur qui doit jouer ;
- `gagne` est un booléen qui vaut FAUX si la partie n'est pas remportée et VRAI si un des joueurs a gagné. C'est ce booléen qui permettra de mettre fin au jeu dès lors qu'il vaut VRAI.

Entrées et sorties des différentes fonctions

Les fonctions principales étant identifiées, il faut maintenant réfléchir aux entrées et aux sorties de chacune d'elle : c'est-à-dire aux paramètres de la fonction (de quoi a-t-elle besoin pour travailler ?) et aux valeurs renvoyées par la fonction (que doit-elle renvoyer ?)

La fonction `afficher_plateau`

La fonction `afficher_plateau` a besoin de connaître le contenu du tableau `jeu` pour afficher le plateau de jeu à l'écran. Et c'est tout ! Ce sera donc son seul paramètre et cette fonction ne renvoie rien (elle affiche quelque chose uniquement).

```python
def afficher_plateau(jeu):
    """Affiche le contenu du tableau jeu."""
    # sera à compléter !
```

Python

Exemple :

```
>>> jeu = [['X', 'O', '.'], ['.', '.', '.'], ['.', '.', 'X']]
>>> afficher_plateau(jeu)
['X', 'O', '.']
['.', '.', '.']
['.', '.', 'X']
```

Python

Exemple (amélioration possible de l'affichage) :

```
>>> jeu = [['X', 'O', '.'], ['.', '.', '.'], ['.', '.', 'X']]
>>> afficher_plateau(jeu)
X | O |
--+---+--
  |   |
--+---+--
  |   | X
```

Python
Voir l'aide n°1/2
Voir l'aide n°2/2

La fonction `jouer`

La fonction `jouer` a pour rôle de demander à l'utilisateur dans quelle ligne et quelle colonne il veut jouer, puis elle doit actualiser et donc modifier l'état du tableau `jeu` en conséquence. Cette fonction a donc a besoin de ce tableau en paramètre. De plus, cette fonction a besoin de connaître le contenu de la variable `joueur` (de type `str`) pour écrire soit `'X'` soit `'O'` dans le tableau `jeu`. Elle possède donc deux paramètres (`jeu` et `joueur`) et ne renvoie rien.

```
def jouer(jeu, joueur):
    """Demande à l'utilisateur la case à jouer et met à jour le tableau jeu."""
    # sera à compléter !
```

Python

Exemple :

```
>>> jeu = [['X', 'O', '.'], ['.', '.', '.'], ['.', '.', 'X']]
>>> jouer(jeu, 'O')  # on suppose que le joueur choisit de jouer dans la case centrale
Quelle colonne veux-tu jouer ? (0, 1 ou 2) : 1
Quelle ligne veux-tu jouer ? (0, 1 ou 2) : 1
>>> jeu
[['X', 'O', '.'], ['.', 'O', '.'], ['.', '.', 'X']]
```

Python
Voir l'aide n°1/2
Voir l'aide n°2/2

La fonction `verifier_victoire`

La fonction `verifier_victoire` a besoin de connaître l'état du plateau donc le tableau `jeu` pour déterminer si la partie est gagnée. Et c'est tout, ce sera son seul paramètre. Cette fonction doit indiquer si la partie est gagnée ou non, elle renverra donc un booléen (VRAI si la partie est gagnée et FAUX sinon).

```
def verifier_victoire(jeu):
    """Renvoie True si la partie est gagnée et False sinon."""
    # sera à compléter !
```

Python

Exemples :

```
>>> jeu = [['X', 'X', 'X'], ['O', '.', '.'], ['O', '.', '.']]
>>> verifier_victoire(jeu)
True
>>> jeu = [['O', '.', '.'], ['.', 'X', '.'], ['X', '.', '.']]
>>> verifier_victoire(jeu)
False
```

Python

Voir l'aide n°1/3

-
-

Voir l'aide n°2/3
Voir l'aide n°3/3

La fonction `changer_joueur`

La fonction `changer_joueur` a besoin de connaître qui vient de jouer pour passer à l'autre joueur. Et c'est tout, donc elle n'a besoin que du paramètre `joueur`. Cette fonction doit aussi modifier le contenu de la variable `joueur` donc elle doit renvoyer une valeur : la chaîne de caractères `'X'` ou la chaîne de caractères `'O'`.

```
def changer_joueur(joueur):
    """Renvoie la chaîne 'X' si joueur vaut 'O' et la chaîne 'O' si joueur vaut 'X'."""
    # sera à compléter !
```

Python

Voir l'aide n°1/1

Écriture de l'algorithme

L'algorithme final du jeu peut alors s'écrire de la façon suivante, dans lequel on utilise les variables `jeu`, `joueur`, `gagne`, ainsi que les fonctions `afficher_plateau()`, `jouer()`, `verifier_victoire()`, `changer_joueur()` définies précédemment.

```
afficher_plateau(jeu)
```

```
joueur ← 'X'                              # le joueur 'X' commence
(arbitraire)
gagne ← FAUX                              # il n'y a pas de gagnant
au départ
Tant que non gagne faire                  # tant qu'il n'y a pas de
gagnant
    Afficher "Au tour de", joueur
    jouer(jeu, joueur)                    # Le joueur propose sa case
et maj du plateau
    afficher_plateau(jeu)
    si verifier_victoire(jeu)              # Si verifie_victoire(jeu)
renvoie VRAI
    alors
        gagne ← VRAI                      # le booleen gagne prend la
valeur VRAI (ce qui stoppera la boucle while)
        Afficher "Le joueur", joueur, "a gagné !"
    sinon
        joueur ← changer_joueur(joueur) # sinon on passe au joueur
suivant
```

À vous de jouer !

Démarche attendue

La démarche de projet est toujours sensiblement la même, quel que soit le projet :

Étape 1 : Répartition des tâches principales = qui fait quoi ?

Répartissez-vous les différentes fonctions `afficher_plateau()`, `jouer()`, `verifier_victoire()`, `changer_joueur()`

Étape 2 : Travail **individuel** pour implémenter (= programmer) la (ou les) fonction(s) qui vous est (sont) attribuée(s). Voici les étapes à suivre pour implémenter une fonction :
- Commencez par bien comprendre ce qu'elle doit faire, en donnant différents exemples d'appels et ce qu'ils doivent produire/renvoyer
- Programmez la fonction (1ère tentative)
- Testez la fonction en faisant des appels qui couvrent les différents cas de figure
- si les tests sont concluants, passer à la suite
- sinon, analysez les erreurs/problèmes et revenez au deuxième point pour procéder aux corrections

 Cette phase de tests/corrections est tout à fait normale et importante ! Il faudra la présenter dans le compte-rendu (quel(s) étai(en)t le(s) problème(s) ? quelle(s) solution(s) avez-vous trouvée(s) ? ...) --> donc il faut garder une trace de tout cela !

N'hésitez pas à regarder les aides pour chaque fonction, mais seulement après avoir réfléchi suffisamment de temps et testez plusieurs choses.

Étape 3 : Écriture du programme principal : traduire l'algorithme du jeu dans le langage précédent

- le premier à avoir terminé l'étape 2, peut se charger de cela, aidé par les autres si nécessaire lorsqu'ils ont terminé leurs tâches
- le programme principal peut tout à fait être écrit sans que les fonctions principales ne soient programmées ! (évidemment, le programme ne fonctionnera pas dans ce cas, mais on peut déjà l'écrire)

Étape 4 : Tests du programme principal

- faire des essais de parties pour vérifier si cela fonctionne
- identifier les problèmes restants (il y en a car l'algorithme proposé n'est pas tout à fait satisfaisant, volontairement !) :
    - en se plaçant dans des cas particuliers
    - en essayant de faire planter le programme

Étape 5 : Correction et amélioration du programme principal (et éventuellement des fonctions principales) pour qu'il n'y ait plus de problèmes.

Sujet 22 : projet long Administrateur réseau

- Déposer dans l'espace indiqué par Le professeur un script Python contenant les réponses aux deux exercices en le nommant selon le format NOM1_NOM2_Sujet22.py
- Insérer un commentaire avec les noms des deux membres du binôme au début du script.
- Respecter les consignes données dans chaque exercice.
- Commenter les parties du code qui ne seraient pas immédiatement compréhensibles à la lecture.
- Fournir pour chaque question des tests permettant de vérifier la validité du code en les plaçant entre triple guillemets.
- Préparer une présentation orale de son travail devant le groupe (3 minutes).

Sujet :

Objectif

Vous êtes l'administrateur réseau d'un lycée et êtes en charge de fournir un identifiant et un mot de passe à tous les lycéens afin qu'ils aient accès à leur compte sur le réseau du lycée.

Le chef d'établissement vous fournit un fichier CSV appelé `liste_eleves.csv` contenant les noms et prénoms de tous les élèves du lycée et vous charge de créer un nouveau fichier CSV contenant les noms, prénoms, identifiants et mots de passe de tous les élèves

Le fichier CSV à télécharger contenant les noms et prénoms de tous les élèves : est ici : https://info-mounier.fr/premiere_nsi/projets/data/liste_eleves.csv

Cahier des charges

L'**identifiant** est formé de la façon suivante : première lettre du prénom suivie du nom de famille, le tout en minuscules. Par exemple, l'élève Roxane Dujardin a pour identifiant `rdujardin`. Il est bien sûr impossible que deux élèves aient le même identifiant !

> Que faire si un autre élève s'appelle Richard Dujardin ? (pourra être traité une fois que le reste est fonctionnel)

Pour respecter les recommandations de la CNIL (Mots de passe : une nouvelle recommandation pour maîtriser sa sécurité), les **mots de passe** sont à définir de manière aléatoire et doivent être composés d'au minimum 12 caractères comprenant des majuscules, des minuscules, des chiffres et des caractères spéciaux à choisir dans une liste d'au moins 37 caractères spéciaux possibles.