Nom du premier membre du groupe : MOSAAD Chehab
Nom du deuxième membre du groupe : ELKHATEEB Mariam
Classe : 1A STRI

DM N°1 (2023-2024) :

Programmation en Python :

Sujet :

Exercice 1 :

Écrivez en Python la fonction intersection_2_listes qui, étant donné deux listes d'entiers L1 et L2 triées en ordre croissant, renvoie la liste des éléments appartenant à la fois à L1 et à L2.
Soit L une liste de listes d'entiers, chacune triée en ordre croissant. Écrivez en Python la fonction intersection telle qu'intersection(L) renvoie la liste triée des entiers appartenant à chacune des listes de L.

Exercice 2 :

- Implémenter en python la structure de données « Pile ».
- Écrivez un programme qui convertit un nombre décimal en sa représentation binaire en utilisant une pile.

Exercice 3 :

Les mots de passe permettent de protéger les comptes des utilisateurs contre les accès non autorisés, pouvant être tentés par d'autres personnes que leurs propriétaires. Ils doivent donc être difficiles à deviner. En général, un bon mot de passe est une chaine de caractères longue formée d'une combinaison de :

- Lettres minuscules
- Lettres majuscules
- Chiffres
- Caractères spéciaux

Écrivez une fonction python qui permet de tester la validité du mot de passe qui lui est transmis en paramètre.
Écrivez un programme python qui permet de tester la fonction précédente.

Exercice 4 :

Une page web est un fichier texte dont le contenu est décrit, structuré avec le langage HTML5 et ses balises :

- Écrivez une fonction qui permet de vérifier si un code HTML donné à des balises correctement équilibrées : vous devez vérifier que les balises d'ouverture et de fermeture correspondent correctement.
- Écrivez un programme Python qui lit un fichier HTML et retourne le nombre d'occurrences de chaque balise HTML présente dans le fichier.

Exercice 5 :

Écrivez un programme Python qui utilise les modules os et datetime pour lister les fichiers se trouvant dans un répertoire donné qui ont été créés après une certaine date limite. Le programme vous demandera de saisir au clavier le nom du répertoire que vous souhaitez explorer ainsi que la date limite à partir de laquelle vous souhaitez rechercher les fichiers. Il affichera ensuite la liste des fichiers qui répondent à ces critères.

Sommaire :

Introduction :

Le langage Python est un langage de programmation très populaire et puissant, qui permet de réaliser de nombreuses applications dans des domaines variés, tels que le web, la science, le jeu, ou encore l'intelligence artificielle. Il se caractérise par sa syntaxe simple et élégante, sa richesse en structures de données, sa facilité d'apprentissage, et sa grande modularité grâce à ses nombreux modules et bibliothèques.

Dans ce DM, nous allons mettre en pratique nos connaissances de Python en réalisant cinq exercices qui couvrent différents aspects du langage, tels que les fonctions, les structures de données, les modules, les fichiers, et le traitement du langage HTML. Ces exercices nous permettront de renforcer nos compétences en programmation, de résoudre des problèmes concrets, et de découvrir de nouvelles fonctionnalités de Python.

Solution de l'exercice 1 :

- **Pour la première fonction :**
  - **L'analyse :**

    Le problème à résoudre est de trouver les éléments communs entre deux listes d'entiers triées en ordre croissant.

    Les principales difficultés sont de parcourir les deux listes de manière efficace, sans répéter les éléments, et de renvoyer une liste triée.

    La solution adoptée pour surmonter ces difficultés est d'utiliser deux indices, un pour chaque liste, et de les incrémenter selon la comparaison des éléments. On ajoute à la liste de résultat les éléments égaux, et on avance dans la liste qui contient l'élément le plus petit.

  - **Les algorithmes à implémenter :**
    - Initialiser une liste vide pour stocker le résultat
    - Initialiser deux indices i et j à 0
    - Tant que i et j sont inférieurs aux longueurs de L1 et L2 respectivement :
      - Si L1[i] est égal à L2[j] :
        - Ajouter L1[i] au résultat
        - Incrémenter i et j
      - Sinon, si L1[i] est inférieur à L2[j] :
        - Incrémenter i
      - Sinon :
        - Incrémenter j

- Renvoyer le résultat

- **Pour la deuxième fonction :**
  - **L'analyse :**

    Le problème à résoudre est de trouver les éléments communs entre plusieurs listes d'entiers triées en ordre croissant.

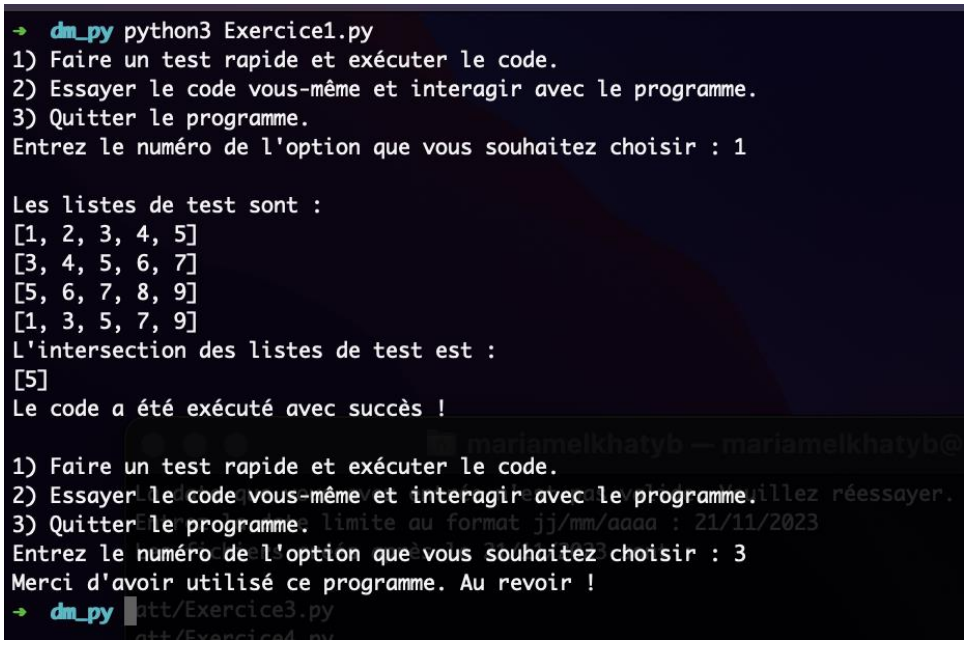
    Les principales difficultés sont de généraliser le cas de deux listes à un nombre arbitraire de listes, et de renvoyer une liste triée.

    La solution adoptée pour surmonter ces difficultés est d'utiliser la fonction intersection_2_listes définie précédemment, et de l'appliquer successivement à chaque liste de L. On utilise une variable intermédiaire pour stocker le résultat partiel de l'intersection.

  - **Les algorithmes à implémenter :**

    - Si L est vide, renvoyer une liste vide
    - Sinon, initialiser le résultat avec la première liste de L
    - Pour chaque liste de L à partir de la deuxième :
      - Calculer l'intersection entre le résultat et la liste courante en utilisant la fonction intersection_2_listes
      - Affecter le résultat de l'intersection à la variable résultat
    - Renvoyer le résultat

- **Les résultats renvoyés par le code :**
  - **Cas sans erreur :**



Capture 1 : Exécution du Programme de Test Rapide

```
dm_py python3 Exercice1.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 2

Entrez le nombre de listes que vous souhaitez comparer : 2
Entrez les éléments de la liste 1 séparés par des espaces : 1 2 3 4 5 6
Entrez les éléments de la liste 2 séparés par des espaces : 2 7 8 9 6
L'intersection des listes que vous avez saisies est : [2, 6]

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
dm_py
```

Capture 2 : Interaction Personnalisée avec le Programme

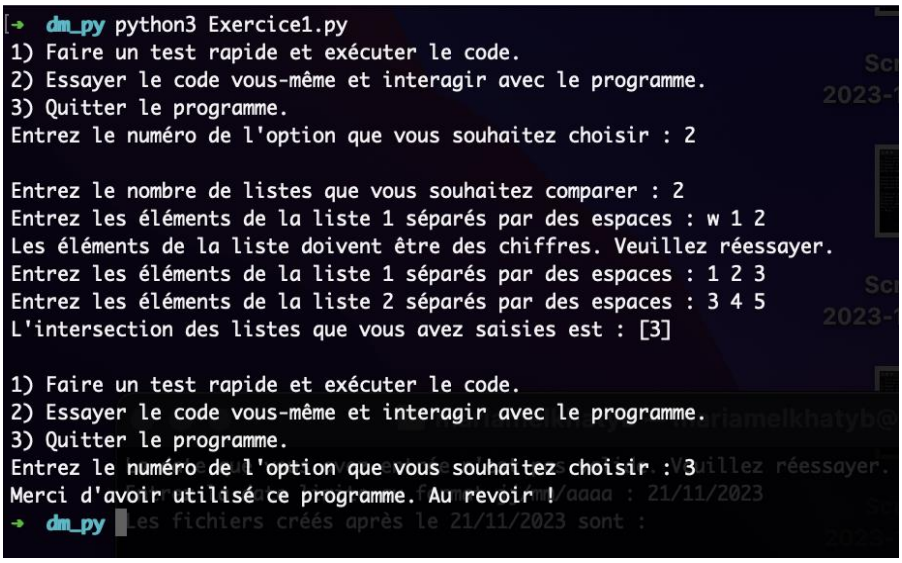
- **<u>Cas d'erreur :</u>**

```
dm_py python3 Exercice1.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : e
Option invalide, veuillez réessayer.

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
dm_py
```

Capture 1 : Saisie Incorrecte du Nombre d'Options

```
dm_py python3 Exercice1.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 2

Entrez le nombre de listes que vous souhaitez comparer : e
Vous devez entrer un seul nombre entier. Veuillez réessayer.

Entrez le nombre de listes que vous souhaitez comparer : 2
Entrez les éléments de la liste 1 séparés par des espaces : 1 2 3
Entrez les éléments de la liste 2 séparés par des espaces : 3 5 6
L'intersection des listes que vous avez saisies est : [3]

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
dm_py
```

Capture 2 : Saisie Incorrecte du Nombre de Listes



Capture 3 : Erreur lors de la Saisie de la Liste

- **La répartition du travail :**
  Nous avons réalisé le travail en collaboration. Nous avons réparti les tâches de manière équitable et nous avons contribué à chaque étape de l'exercice, donc **50%** chacun.

Solution de l'exercice 2 :

- **L'analyse :**

Le problème à résoudre est de créer une structure de données "Pile", qui peut empiler des chiffres et des caractères, en Python, et de l'utiliser pour convertir un nombre décimal en sa représentation binaire.

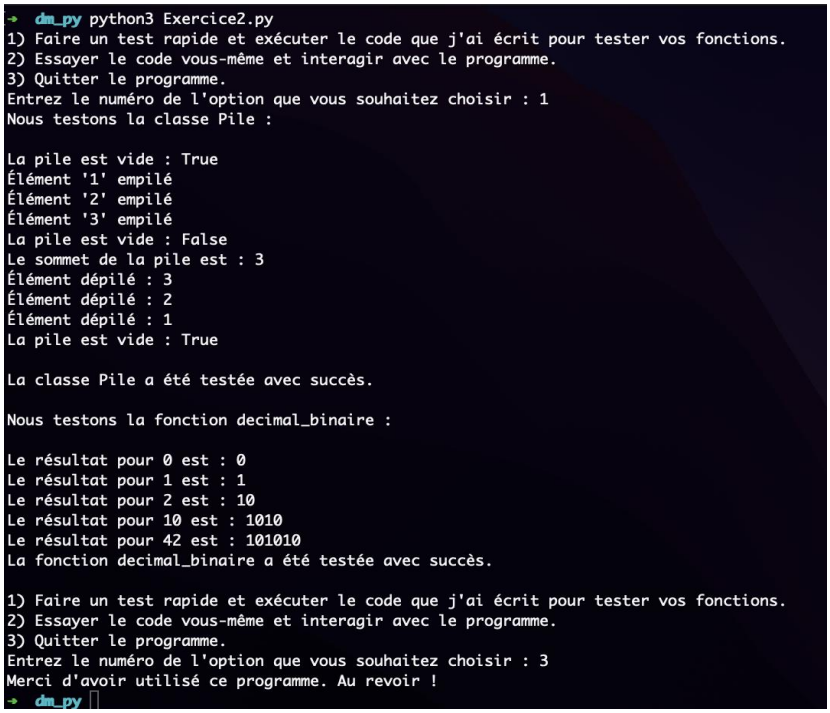
Les principales difficultés sont de respecter les propriétés d'une pile, qui est une structure de données de type LIFO (Last In First Out), c'est-à-dire que le dernier élément ajouté est le premier à être retiré, et de trouver un algorithme efficace pour la conversion décimale-binaire.

La solution adoptée pour surmonter ces difficultés est d'utiliser une liste Python comme support de la pile, et de définir des méthodes pour empiler, dépiler, et vérifier si la pile est vide. Pour la conversion, on utilise le fait qu'un nombre décimal peut s'écrire comme une somme de puissances de 2, et qu'on peut obtenir le bit correspondant à chaque puissance en effectuant des divisions successives par 2.

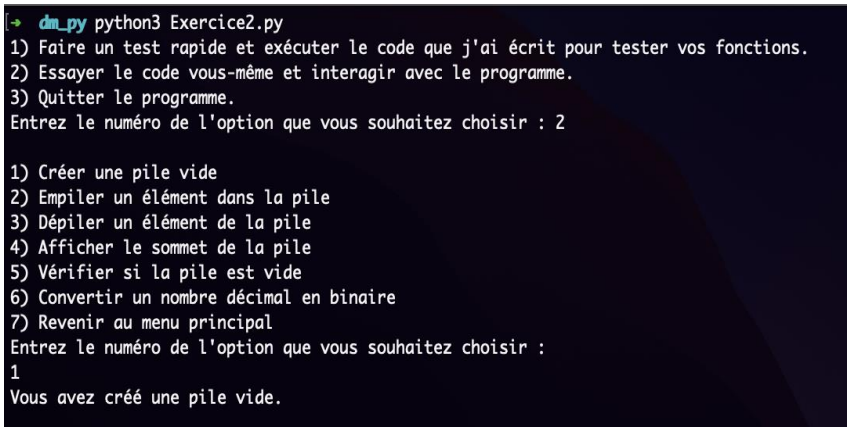
- **Les algorithmes à implémenter :**

  - Créer une pile vide pour stocker les restes de la division par 2
  - Donner un nombre décimal à convertir en binaire
  - Tant que le nombre décimal n'est pas nul :
    - Calculer le reste de la division du nombre décimal par 2
    - Empiler le reste dans la pile
    - Diviser le nombre décimal par 2 et arrondir à l'entier inférieur
  - Initialiser une chaîne vide pour stocker le résultat
  - Tant que la pile n'est pas vide :
    - Dépiler un élément de la pile et le concaténer à la chaîne
  - Renvoyer la chaîne comme la représentation binaire du nombre décimal

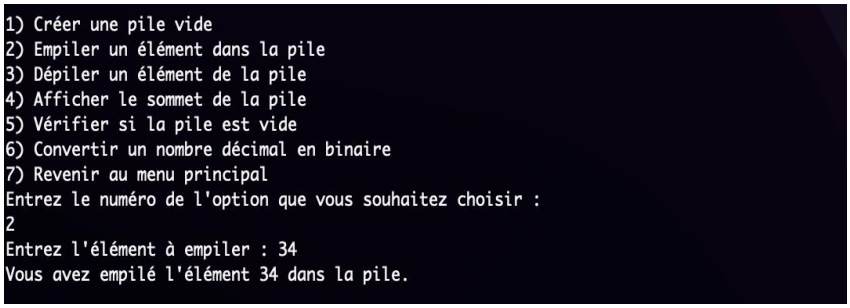
- **<u>Les résultats renvoyés par le code :</u>**



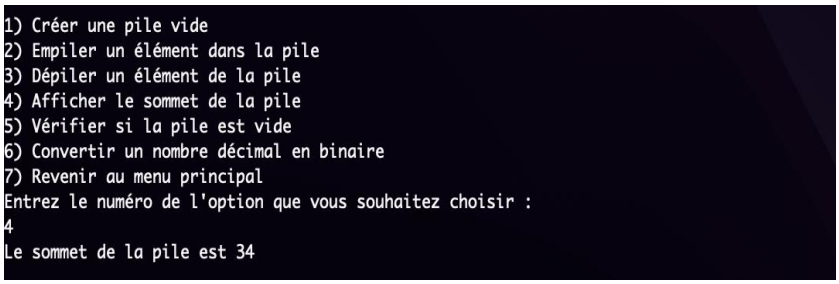
Capture 1 : Exécution du Programme de Test Rapide



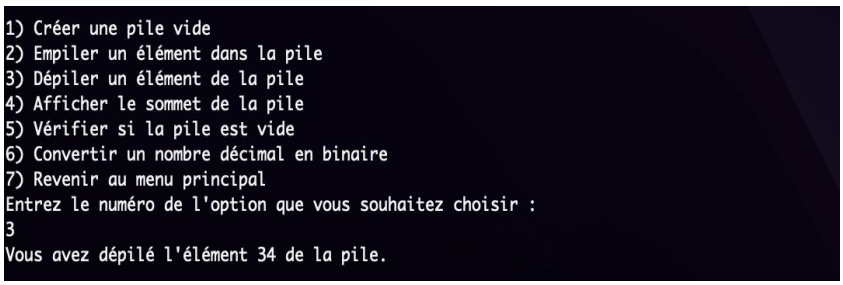
Capture 2 : Création d'une pile vide



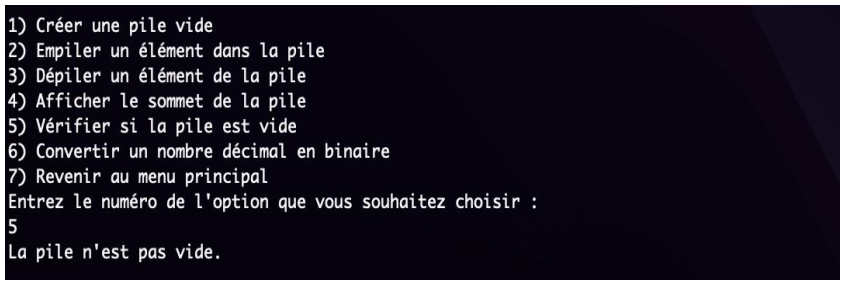
Capture 3 : Empilage d'un élément dans la pile



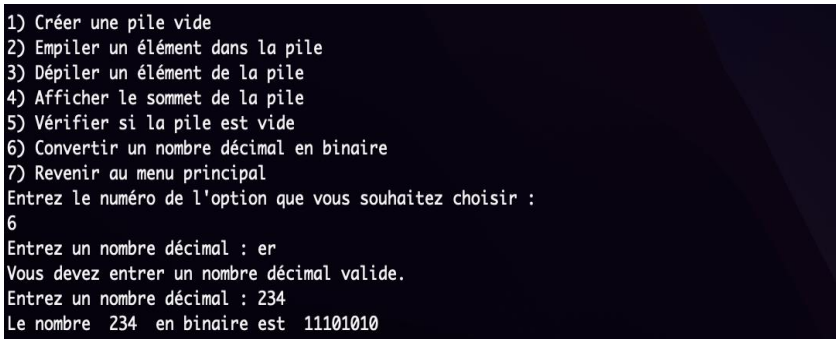
Capture 4 : Affichage du sommet de la pile



Capture 5 : Dépilage d'un élément de la pile



Capture 6 : Vérification si la pile est vide



Capture 7 : Conversion d'un nombre décimal en binaire avec un exemple valide et un exemple invalide

- **La répartition du travail :**
  Nous avons réalisé le travail en collaboration. Nous avons réparti les tâches de manière équitable et nous avons contribué à chaque étape de l'exercice, donc **50%** chacun.

Solution de l'exercice 3 :

- **L'analyse :**

  Le problème à résoudre est de vérifier si un mot de passe est valide selon certains critères de sécurité, tels que la longueur et la diversité des caractères utilisés.

  Les principales difficultés sont de parcourir le mot de passe et de compter le nombre de caractères de chaque catégorie, et de renvoyer un booléen indiquant la validité du mot de passe.

  La solution adoptée pour surmonter ces difficultés est d'utiliser des constantes pour définir les caractères minuscules, majuscules, chiffres et spéciaux, et de les comparer avec chaque caractère du mot de passe. On utilise également des variables pour compter le nombre de caractères de chaque catégorie, et on vérifie à la fin si le mot de passe respecte les conditions de validité.

- **Les algorithmes à implémenter :**

  - Définir les constantes pour les caractères minuscules, majuscules, chiffres et spéciaux
  - Définir la fonction qui teste la validité du mot de passe
    - Donner un mot de passe à tester
    - Initialiser les variables pour compter le nombre de caractères de chaque catégorie
    - Parcourir le mot de passe caractère par caractère
      - Si le caractère est une lettre minuscule, incrémenter le compteur correspondant
      - Si le caractère est une lettre majuscule, incrémenter le compteur correspondant
      - Si le caractère est un chiffre, incrémenter le compteur correspondant
      - Si le caractère est un caractère spécial, incrémenter le compteur correspondant
      - Sinon, le caractère n'est pas valide, renvoyer False
    - Vérifier si le mot de passe respecte les conditions de validité
      - Le mot de passe doit avoir au moins 8 caractères

- Le mot de passe doit avoir au moins un caractère de chaque catégorie
- Si les conditions sont remplies, renvoyer True
- Sinon, renvoyer False

- **Les résultats renvoyés par le code :**
  - **Cas de validation de mot de passe :**

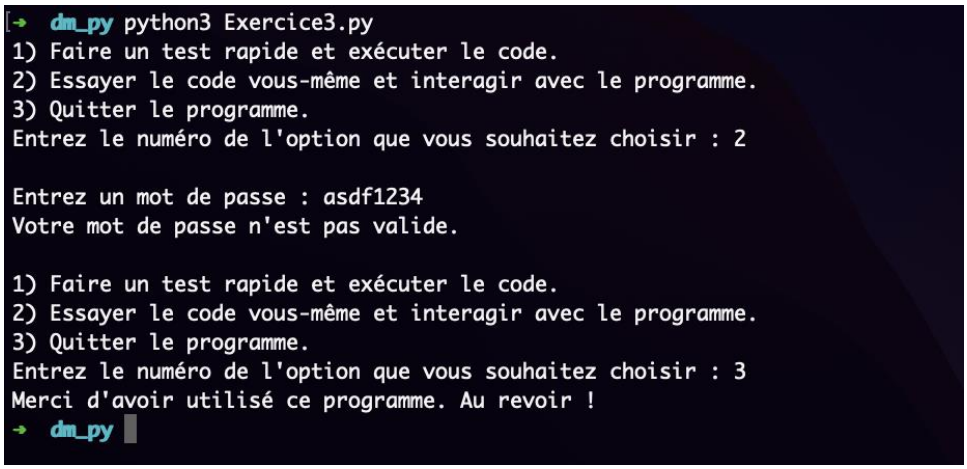
```
➜  dm_py python3 Exercice3.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 1

Les mots de passe de test sont :
azerty
Azerty123
Azerty@123
Azerty@123!

La validité des mots de passe de test est :
False
False
True
True
Le code a été exécuté avec succès !

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
➜  dm_py
```

Capture 1 : Exécution du Programme de Test Rapide

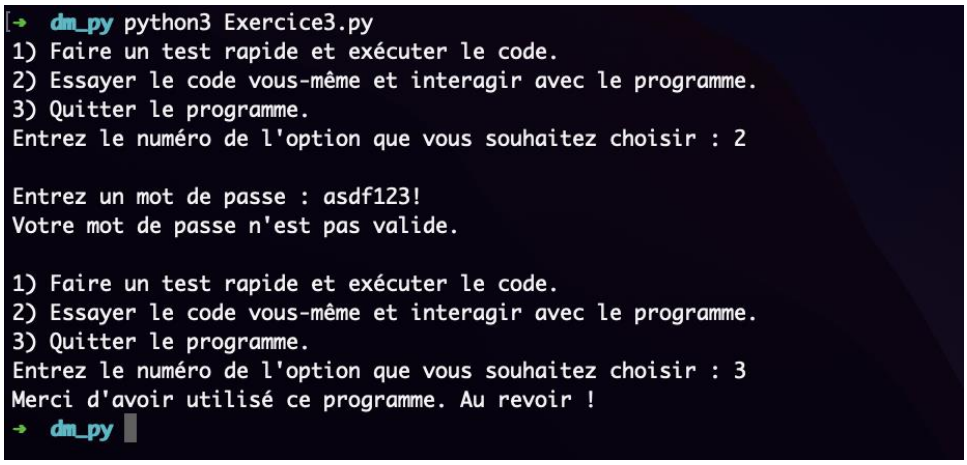
```
➜  dm_py python3 Exercice3.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 2

Entrez un mot de passe : Abngh12!
Votre mot de passe est valide.

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
➜  dm_py
```

Capture 2 : Saisie d'un Mot de Passe Conforme

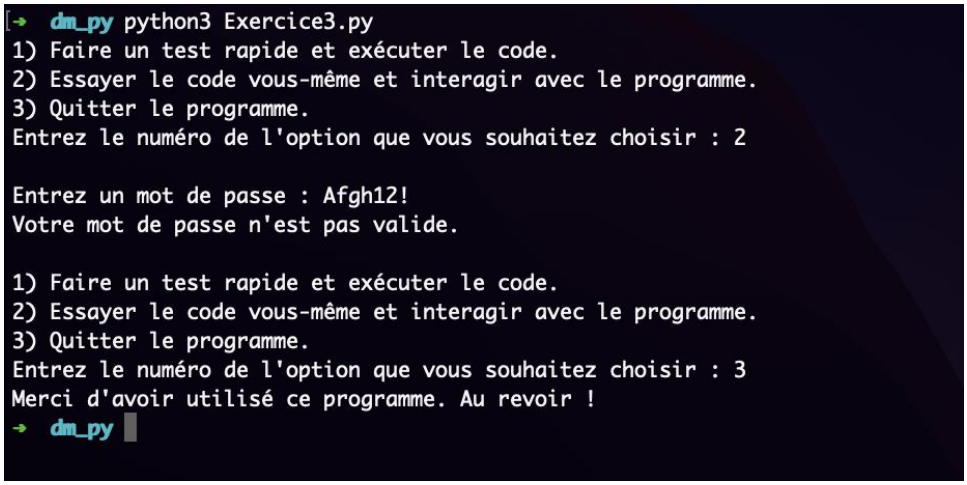
- **Cas d'erreur de validation de mot de passe :**



Capture 1 : Mot de Passe sans Majuscules ni Caractères Spéciaux



Capture 2 : Mot de Passe sans Majuscules



Capture 3 : Mot de Passe Insuffisamment Long

- **La répartition du travail :**
  Nous avons réalisé le travail en collaboration. Nous avons réparti les tâches de manière équitable et nous avons contribué à chaque étape de l'exercice, donc **50%** chacun.

Solution de l'exercice 4 :

- **L'analyse :**
  Le problème à résoudre est de manipuler du code HTML, qui est un langage de balisage utilisé pour décrire la structure et le contenu d'une page web.
  Les principales difficultés sont de parcourir le code HTML et de détecter les balises d'ouverture et de fermeture, et de vérifier si elles sont correctement équilibrées, c'est-à-dire qu'il n'y a pas de balise orpheline ou mal fermée. Il faut également compter le nombre d'occurrences de chaque balise présente dans le code HTML.
  La solution adoptée pour surmonter ces difficultés est d'utiliser une structure de données de type pile, qui permet de stocker les balises d'ouverture rencontrées et de les comparer avec les balises de fermeture correspondantes. On utilise également un dictionnaire pour stocker le nombre d'occurrences de chaque balise.

- **Les algorithmes à implémenter :**

  - Définir la fonction qui vérifie si un code HTML donné a des balises correctement équilibrées
    - Donner un code HTML à vérifier
    - Créer une pile vide pour stocker les balises d'ouverture
    - Parcourir le code HTML caractère par caractère
      - Si on rencontre le caractère '<', on commence une balise
        - On avance jusqu'au caractère '>'
        - On récupère le nom de la balise entre les caractères '<' et '>'
        - Si on a bien trouvé le caractère '>', on traite la balise
          - Si la balise commence par '/', c'est une balise de fermeture
            - On retire le '/' du nom de la balise

- Si la pile est vide, il y a une balise de fermeture sans balise d'ouverture correspondante, le code HTML n'est pas équilibré, on renvoie False
- Sinon, on dépile la dernière balise d'ouverture de la pile
- Si la balise de fermeture ne correspond pas à la balise d'ouverture, le code HTML n'est pas équilibré, on renvoie False

o Sinon, si la balise ne se termine pas par '/', c'est une balise d'ouverture
- On empile la balise d'ouverture dans la pile

o Sinon, c'est une balise auto-fermante, on ne fait rien
- On avance au caractère suivant

o À la fin du parcours, si la pile est vide, le code HTML est équilibré, on renvoie True
o Sinon, il reste des balises d'ouverture sans balise de fermeture correspondante, le code HTML n'est pas équilibré, on renvoie False

- Définir la fonction qui compte le nombre d'occurrences de chaque balise HTML présente dans un fichier
  - Donner le nom du fichier HTML à analyser
  - Ouvrir le fichier HTML en mode lecture
  - Lire le contenu du fichier
  - Créer un dictionnaire vide pour stocker le nombre d'occurrences de chaque balise
  - Parcourir le code HTML caractère par caractère
    - Si on rencontre le caractère '<', on commence une balise
      - On avance jusqu'au caractère '>'
      - On récupère le nom de la balise entre les caractères '<' et '>'
      - Si on a bien trouvé le caractère '>', on traite la balise
        - Si la balise commence par '/', c'est une balise de fermeture, on ne la compte pas
        - Sinon, c'est une balise d'ouverture ou auto-fermante, on la compte
          - Si la balise se termine par '/', on retire le '/' du nom de la balise
          - Si la balise existe déjà dans le dictionnaire, on incrémente son nombre d'occurrences
          - Sinon, on crée une nouvelle entrée dans le dictionnaire avec le nombre d'occurrences à 1
    - On avance au caractère suivant
  - Renvoyer le dictionnaire des occurrences
- Écrire un programme qui utilise la fonction précédente
  - Demander à l'utilisateur d'entrer le nom du fichier HTML à analyser
  - Appeler la fonction compter_balises avec le nom du fichier entré
  - Afficher le résultat sous forme de dictionnaire

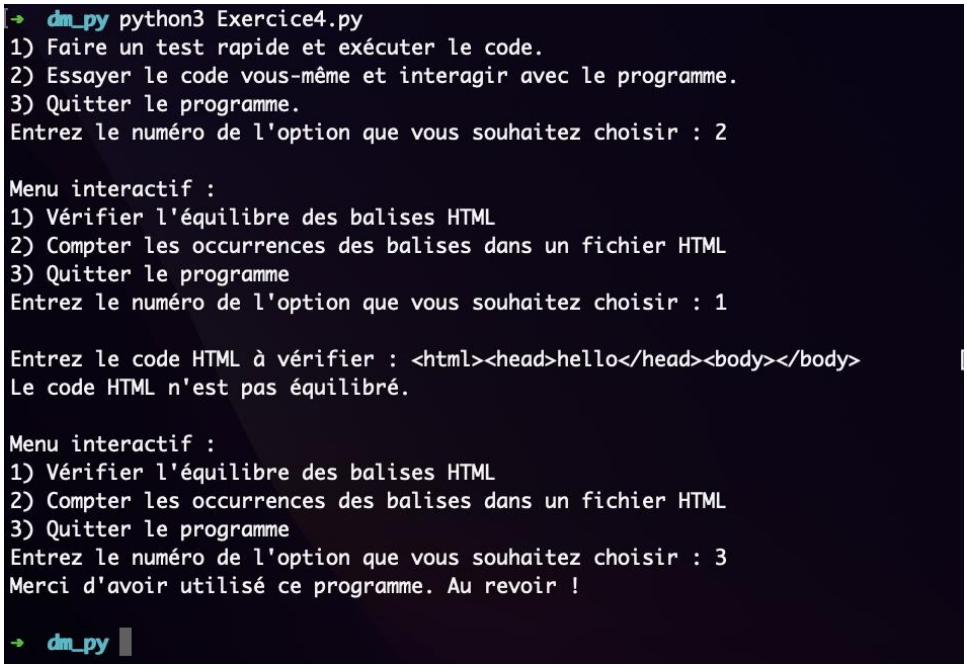
- **<u>Les résultats renvoyés par le code :</u>**

```
➜  dm_py python3 Exercice4.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 1

L'exemple de code de ce test est :  <html><head></head><body></body></html>
Test balises_equilibrees réussi pour un code équilibré.

L'exemple de code de ce test est : <html><head></body></html>
Test balises_equilibrees réussi pour un code non équilibré.

L'exemple de fichier de ce test est : test.html
les resultats sont :  {'html': 1, 'head': 1, 'body': 1}
Test compter_balises réussi.

1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !
➜  dm_py
```

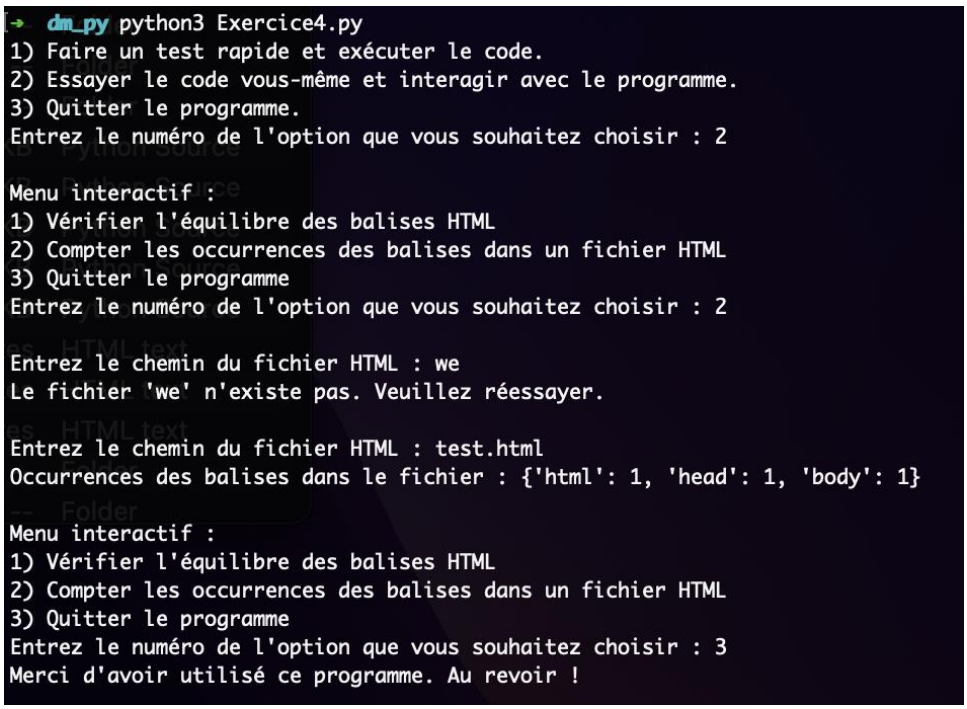
Capture 1 : Exécution du Programme de Test Rapide

```
➜  dm_py python3 Exercice4.py
1) Faire un test rapide et exécuter le code.
2) Essayer le code vous-même et interagir avec le programme.
3) Quitter le programme.
Entrez le numéro de l'option que vous souhaitez choisir : 2

Menu interactif :
1) Vérifier l'équilibre des balises HTML
2) Compter les occurrences des balises dans un fichier HTML
3) Quitter le programme
Entrez le numéro de l'option que vous souhaitez choisir : 1

Entrez le code HTML à vérifier : <html><head>hello</head><body></body></html>
Le code HTML est équilibré.

Menu interactif :
1) Vérifier l'équilibre des balises HTML
2) Compter les occurrences des balises dans un fichier HTML
3) Quitter le programme
Entrez le numéro de l'option que vous souhaitez choisir : 3
Merci d'avoir utilisé ce programme. Au revoir !

➜  dm_py
```

Capture 2 : Test d'équilibre des balises HTML avec un exemple équilibré



Capture 3 : Test d'équilibre des balises HTML avec un exemple non équilibré



Capture 4 : Nombre occurrences des balises dans un fichier HTML avec des saisies incorrectes

Capture 5 : Fichier HTML utilisé

- **La répartition du travail :**
  Nous avons réalisé le travail en collaboration. Nous avons réparti les tâches de manière équitable et nous avons contribué à chaque étape de l'exercice, donc **50%** chacun.

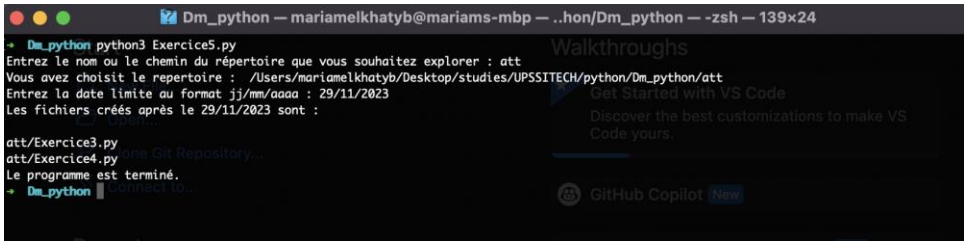
Solution de l'exercice 5 :

- **L'analyse :**
  Le problème à résoudre est de lister les fichiers d'un répertoire donné qui ont été créés après une certaine date limite.
  Les principales difficultés sont de parcourir le répertoire et de récupérer les informations sur les fichiers, et de comparer les dates de création avec la date limite.
  La solution adoptée pour surmonter ces difficultés est d'utiliser les modules os et datetime de Python, qui fournissent des fonctions pour manipuler les chemins, les fichiers et les dates.

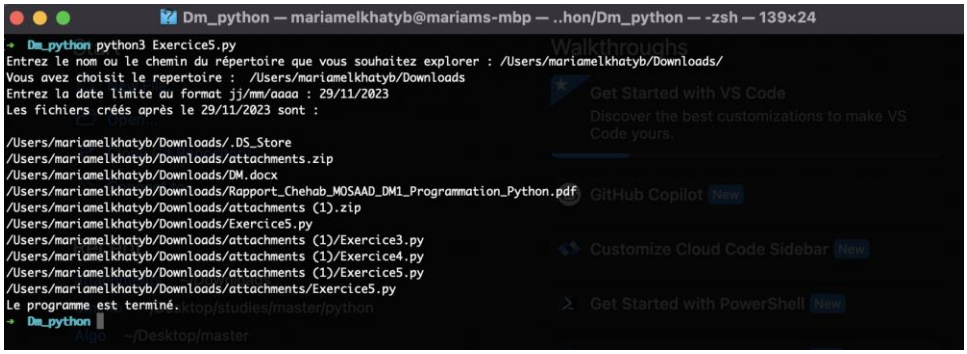
- **Les algorithmes à implémenter :**

  - Importer les modules os et datetime pour manipuler les fichiers et les dates
  - Demander à l'utilisateur de saisir le nom du répertoire à explorer
  - Demander à l'utilisateur de saisir la date limite au format jj/mm/aaaa
  - Convertir la date limite en un objet datetime qui représente la date et l'heure
  - Créer une liste vide pour stocker les fichiers qui répondent aux critères
  - Parcourir le répertoire et ses sous-répertoires avec la fonction os.walk
    - Pour chaque fichier rencontré
      - Récupérer le chemin complet du fichier avec la fonction os.path.join
      - Récupérer la date de création du fichier avec la fonction os.path.getctime
      - Convertir la date de création en un objet datetime avec la fonction datetime.datetime.fromtimestamp
      - Si la date de création est supérieure à la date limite
        - Ajouter le fichier à la liste des fichiers qui répondent aux critères
  - Afficher la liste des fichiers qui répondent aux critères
    - Afficher la date limite au format jj/mm/aaaa avec la méthode datetime.strftime
    - Afficher chaque fichier de la liste avec une boucle for

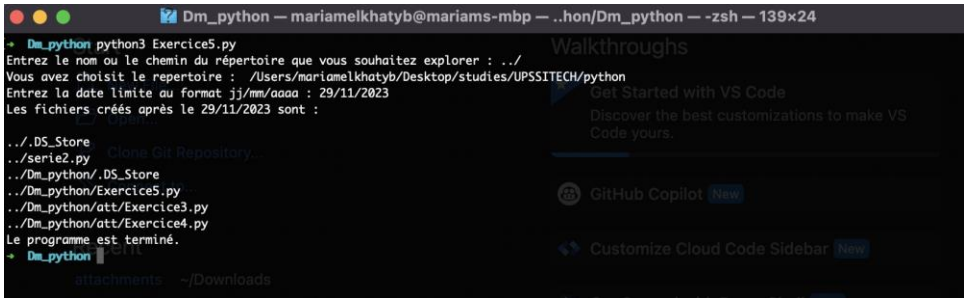
- **Les résultats renvoyés par le code :**
  - **Cas sans erreur :**



Capture 1 : Saisie d'un nom de répertoire dans le même fichier

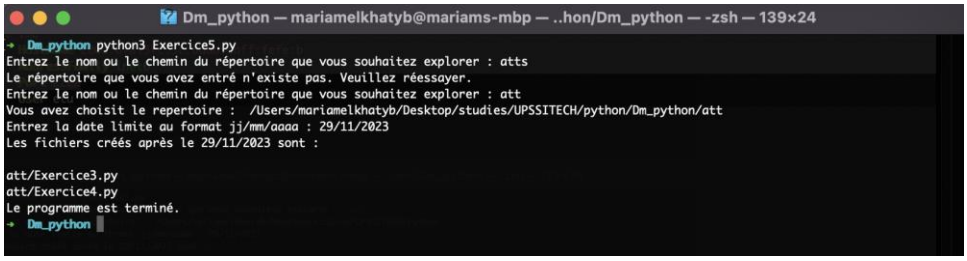


Capture 2 : Saisie du Chemin Absolu de répertoire

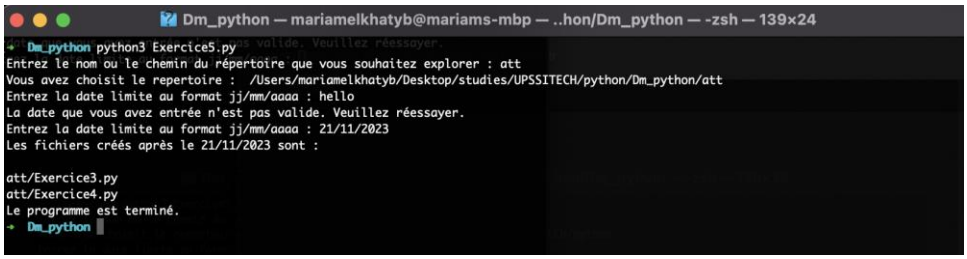


Capture 3 : Saisie du Chemin relatif de répertoire

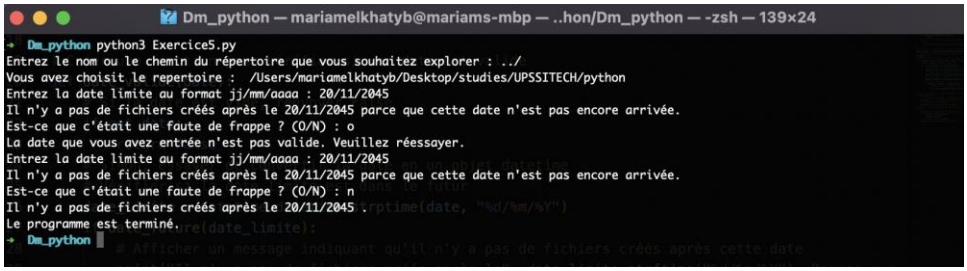
  - **Cas d'erreur :**



Capture 1 : Saisie d'un Nom de Répertoire Inexistant



Capture 2 : Saisie des caractères dans le champ de date



Capture 3 : Saisie d'une date future

- **<u>La répartition du travail :</u>**
  Nous avons réalisé le travail en collaboration. Nous avons réparti les tâches de manière équitable et nous avons contribué à chaque étape de l'exercice, donc **50%** chacun.

<u>Conclusion :</u>

En conclusion, ce DM nous a permis de réviser et d'approfondir nos connaissances de Python, en abordant des sujets variés et intéressants. Nous avons pu utiliser des fonctions pour modulariser notre code, des structures de données pour stocker et manipuler des informations, des modules pour accéder à des fonctionnalités avancées, et des fichiers pour lire et écrire des données. Nous avons également appris à traiter du code HTML, utilisé pour créer des pages web.

Également, ce DM nous a également permis de développer notre méthodologie de programmation, en analysant les problèmes à résoudre, en concevant des algorithmes adaptés, en écrivant du code clair et commenté, et en testant nos solutions. Nous avons ainsi pu améliorer notre logique, notre rigueur, et notre créativité.

Enfin, ce DM nous a apporté une satisfaction personnelle, en nous faisant découvrir de nouvelles possibilités offertes par Python, et en nous confrontant à des défis stimulants. Nous avons pu apprécier la beauté et la puissance de ce langage, ainsi que son utilité dans de nombreux domaines.

Pour améliorer notre travail, nous pourrions ajouter des fonctionnalités supplémentaires à nos programmes, tels que des options de configuration, ou des messages d'erreur. Nous pourrions également optimiser notre code, en utilisant des techniques de programmation plus efficaces, ou en utilisant des modules plus performants. Nous pourrions enfin enrichir notre travail, en explorant d'autres aspects de Python, tels que la programmation orientée objet, la programmation fonctionnelle, ou la programmation concurrente.