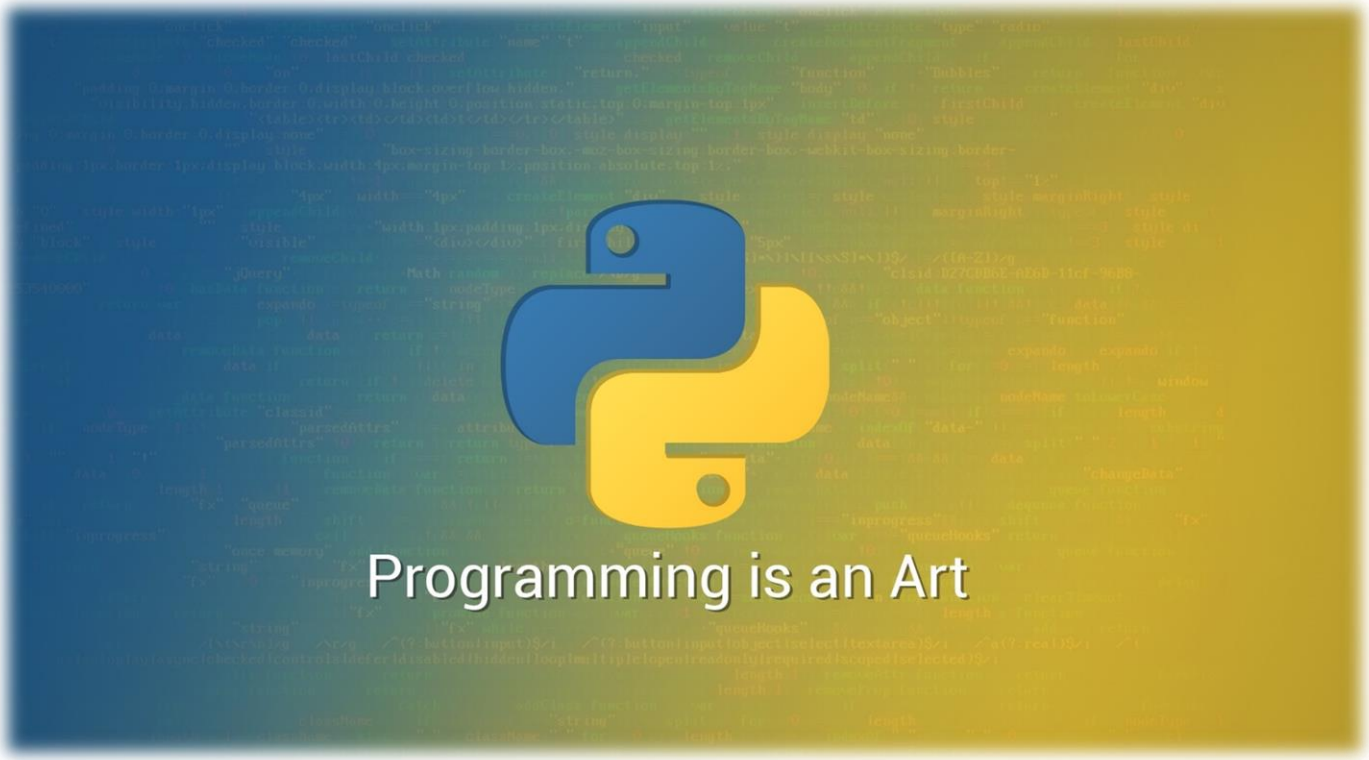
Formation Python par l'exemple : DAEU

DarkSATHI Li

2024 - 2025



NOM : .....

PRENOM : .....

DarkSATHI Li

Sommaire

Les bases

1

Demandez le prénom de l'utilisateur et affichez le message de sortie

Bonjour [Prénom].

Demandez le prénom de l'utilisateur, puis demandez lui son nom de famille et affichez le message de sortie

Bonjour [Prénom] [Nom].

2

Écrivez un code qui affichera la blague

- « Quand est ce que Windows ne bug pas? »

et sur la ligne suivante affichez la réponse

- « Quand l'ordinateur est éteint. »

Essayez de le créer en utilisant une seule. ligne de code.

3

Demandez à l'utilisateur d'entrer deux nombres. Ajoutez les ensemble et affichez la réponse comme

Le total est [répondre].

4

Demandez à l'utilisateur d'entrer trois nombres. Additionnez le premier avec le deuxième nombre puis multipliez cette somme par le troisième. Affichez le répondre comme

La réponse est [répondre].

5

6

Demandez combien de tranches de pizza l'utilisateur à commandé et combien de tranches il a mangé.
Calculez combien de tranches il reste et affichez la réponse à l'utilisateur dans un format convivial.

Demandez à l'utilisateur son nom et son âge. Ajouter 1 à son âge et affichez la sortie

[Nom] lors de votre prochain anniversaire
Vous aurez [nouvel âge].

7

8

Demandez le prix total de la facture du restaurant, puis demandez combien il y a de convives dans le restaurant. Divisez la facture totale par le nombre de convives et indiquez combien chaque personne doit payer.

Ecrire un programme qui demandera un nombre de jours et affichez combien d'heures, minutes et secondes sont dans ce nombre de jours.

9

Il y a 2,204 livres dans un kilogramme. Demandez à utilisateur d'entrer un poids en kilogrammes et affichez la conversion en livres.

10

Demandez à l'utilisateur d'entrer un nombre supérieur à 100, puis d'entrer un nombre inférieur à 10 et dites-lui combien de fois le plus petit nombre entre dans le plus grand en affichant un message dans un format convivial.

11

La structure conditionnelle : if

1

Demandez deux nombres. Si le premier est plus grand que le second, affichez le second nombre en premier puis le premier nombre, sinon affichez le premier nombre en premier et puis le second.

2

Demandez à l'utilisateur d'entrer un nombre qui est inférieur à 20. Si on donne un nombre supérieur ou égal à 20, affichez le message "Trop élevé", sinon affichez "Merci".

3

Demandez à l'utilisateur d'entrer un nombre entre 10 et 20(compris). Si il entre un nombre dans cette plage, affichez le message "Merci à vous", sinon affichez le message "Réponse incorrecte".

4

Demandez à l'utilisateur d'entrer sa couleur préférée. Si il entre "rouge", "ROUGE" ou "Rouge" affichez le message "J'aime le rouge aussi", sinon affichez le message "Je n'aime pas [la couleur], je préfère le rouge".

5

Demandez à l'utilisateur s'il pleut et convertissez sa réponse en minuscules donc peu importe la casse dans laquelle il le tape. Si il répond "oui", demandez si il y a du vent. Si il répond "oui" à cette deuxième question, affichez la réponse "Il y a trop de vent pour un parapluie", sinon affichez le message "Prenez un parapluie". Si la réponse à la première question est "non", affichez la réponse "Profitez de votre journée".

6

Demandez l'âge de l'utilisateur. Si il a 18 ans ou plus, affichez le message "Vous pouvez voter", si il a 17 ans, affichez le message "Vous pouvez apprendre à conduire", si il a 16 ans, affichez le message "Vous pouvez acheter un billet de loterie", si il a moins de 16 ans, affichez le message "Vous pouvez seulement prendre votre vélo".

Demandez à l'utilisateur d'entrer un nombre. Si il est inférieur à 10, affichez le message "Trop bas", si le nombre est entre 10 et 20, afficher "Correct", sinon afficher "Trop élevé".

7

Demandez à l'utilisateur d'entrer 1, 2 ou 3. Si il entre un 1, affichez le message "Merci", si il saisit un 2, affichez "Bien joué", si il entre un 3, affichez "Correct". Si il entre autre chose, affichez "Message d'erreur".

8

Les chaînes de caractères

1

Demandez à l'utilisateur d'entrer son prénom et puis affichez le longueur de son prénom.

Demandez à l'utilisateur d'entrer son prénom, puis demandez-lui d'entrer son nom de famille. Joignez-les avec un espace entre et affichez les ainsi que le nombre de lettres total du nom et prénom.

2

Demandez à l'utilisateur d'entrer son prénom et son nom en minuscules. Changez la casse en casse du titre et joignez-les ensemble. Affichez le résultat final.

3

Demandez à l'utilisateur de taper la première ligne d'une comptine et affichez la longueur de la chaîne. Demandez un entier de départ et un entier de fin, puis affichez juste cette partie du texte(Python commence à compter de 0 et non de 1).

4

Demandez à l'utilisateur de taper n'importe quel mot et l'affichez en majuscules.

5

Demandez à l'utilisateur d'entrer son prénom. Si la longueur de son prénom comporte moins de cinq caractères, demandez lui de saisir son nom de famille et concaténer le prénom et le nom (sans espace) et afficher le résultat en majuscules. Si la longueur du prénom est de cinq caractères ou plus, affichez leur prénom en minuscules.

5

Pig Latin prend la première lettre d'un mot si c'est une consonne, la déplace à la fin du mot et ajoute un "ay". Si un mot commence par une voyelle, il suffit d'ajouter "way" à la fin. Par exemple, pig devient igpay, banana devient ananabay, et aadvark devient aadvarkway. Créez un programme qui demandera à l'utilisateur d'entrer un mot et de le changer en Pig Latin. Assurez-vous que le nouveau mot est affiché en minuscules.

6

Les mathématiques

1

Demandez à l'utilisateur d'entrer un nombre avec beaucoup de décimales. Multipliez ce nombre par deux et affichez la réponse.

Mettez à jour le programme 1 pour qu'il affiche la réponse avec deux décimales.

2

3

Demandez à l'utilisateur d'entrer un nombre entier supérieur à 500. Calculer la racine carrée de ce nombre et affichez le résultat avec deux décimales.

Demandez à l'utilisateur d'entrer le rayon d'un cercle. Affichez l'aire du cercle.

4

5

Demander le rayon et la hauteur d'un cylindre et calculez le volume total arrondi à trois décimales. Affichez le résultat.

Demandez à l'utilisateur d'entrer deux entiers naturels. Utilisez la division des nombres entiers pour diviser le premier chiffre par le second et calculez aussi le reste et affichez la réponse dans un format convivial(par exemple, si on rentre 7 et 2 affichent "7 divisé par 2 a pour quotient entier 3 et pour reste 1.").

6

7

Affichez le menu suivant :

1) Carré
2) Triangle

Si l'utilisateur saisit 1, on doit alors lui demander la longueur d'un des côtés du carré et affichez son aire. Si il sélectionne 2, demandez la base et la hauteur du triangle et affichez son aire. Si il tape autre chose, affichez un message d'erreur.

La boucle POUR : for

1. Demandez à l'utilisateur d'entrer son nom, puis affichez son nom trois fois.

2. Modifiez le programme 1 pour qu'il demande à l'utilisateur d'entrer son nom et un entier naturel, puis affichez son nom le nombre de fois égal à cet entier naturel.

3. Demandez à l'utilisateur d'entrer son nom et affichez chaque lettre du nom sur une ligne distincte.

4. Modifiez de programme 3 pour demander également un entier naturel. Affichez son nom (une lettre à la fois par ligne) et répétez ceci un nombre de fois égal au nombre entier choisi.

5. Demandez à l'utilisateur d'entrer un entier naturel entre 1 et 12 puis affichez la table de multiplication de ce nombre.

6. Demandez un entier naturel inférieur à 50, puis comptez à rebours à partir de 50 jusqu'à ce nombre, en veillant à montrer le nombre choisi dans la sortie.

7. Demandez à l'utilisateur d'entrer son nom et un entier naturel. Si l'entier est inférieur à 10, alors affichez le nom un nombre de fois égal à l'entier choisi; sinon affichez le message "Trop élevé" trois fois.

8. Définissez une variable appelée total initialisée à 0. Demandez à l'utilisateur d'entrer cinq entiers naturels et après chaque entrée, demandez-lui si il veux que cet entier soit conservé. Si c'est le cas, ajoutez le nombre au total. Si il ne veut pas que cet entier soit conservé, ne l'ajoutez pas au total. Après avoir entré les cinq entiers, affichez le total.

Demandez dans quelle ordre l'utilisateur veut compter(croissant ou décroissant). Si il sélectionne croissant, demandez jusqu'à quel entier naturel compter, puis comptez de 1 à ce nombre. Si il sélectionne décroissant, demandez d'entrer un entier naturel inférieur à 20, puis de comptez à rebours de 20 jusqu'à ce nombre. Si il saisi autre chose que croissant ou décroissant, affichez le message "Je ne comprends pas".

9

Demandez combien de personnes l'utilisateur souhaite inviter à sa fête. Si il saisit un nombre inférieur à 10, demandez les noms des invités et après chaque nom affichez " [nom] a été invité". Si il saisit un nombre supérieur ou égal à 10, affichez le message "Trop de personnes".

10

La boucle tant que : while

1

Initialisez la variable total à 0. Alors que total ne doit pas être supérieure à 50, demandez à l'utilisateur d'entrer un entier naturel. Ajoutez ce nombre au total et affichez le message "Le total est [total]". Répétez l'opération tant que total n'est pas supérieure à 50.

Demandez à l'utilisateur d'entrer un entier naturel. Faire cette demande jusqu'à ce que la valeur choisie soit supérieure à 5 et puis affichez le message "Le dernier numéro que vous avez saisi était un [nombre]." et arrêter le programme.

2

3

Demandez à l'utilisateur d'entrer un entier naturel, puis demander d'entrer un autre entier naturel. Ajoutez ces deux nombres et puis demandez si l'utilisateur veut ajouter un autre entier naturel. Si il entre "y", demandez-lui d'entrer cet entier et ajouter jusqu'à ce qu'il ne réponde pas "y". Une fois la boucle arrêtée, affichez le total.

Demandez le nom de quelqu'un que l'utilisateur veut inviter à sa fête. Après cela, affichez le message "[nom] a maintenant été invité" et ajoutez 1 au décompte. Demandez ensuite si il veut inviter quelqu'un d'autre. Continuez à répéter cela jusqu'à ce qu'il ne veuille plus inviter quelqu'un d'autre à sa fête, puis afficher le nombre de personnes qui vont venir à sa fête.

4

Créez une variable appelée compnum et initialisez sa valeur à 50. Demandez à l'utilisateur d'entrer un entier naturel.

- Si la valeur proposée est différente de compnum, précisez à l'utilisateur si elle est trop faible ou trop élevé et demandez une autre proposition.
- Si la valeur est égale à compnum, afficher le message "Bravo, vous avez eu besoin de [count] tentatives".

5

Demandez à l'utilisateur d'entrer un entier naturel entre 10 et 20.Si il saisit une valeur inférieure à 10, affichez le message "Trop bas" et demandez qu'il essaie à nouveau. Si il saisit une valeur supérieure à 20, affichez le message "Trop élevé" et demandez de réessayer. Continuez à répéter ces instructions jusqu'à ce que l'utilisateur entre une valeur qui est entre 10 et 20 puis affichez le message "Merci".

6

À l'aide de la chanson "10 bouteilles vertes", affichez les lignes "Il y a [num] bouteilles vertes accroché au mur et si 1 bouteille verte tombe accidentellement". Posez ensuite la question "combien de bouteilles vertes seront accrochées au mur?" Si l'utilisateur répond correctement, affichez le message "Il y aura [nombre] bouteilles vertes accrochées au mur". Si il ne répond pas correctement, affichez le message "Non, réessayez" jusqu'à ce qu'il réussisse. Quand le nombre de bouteilles vertes est 0, affichez le message "Il n'y a plus de bouteilles vertes accrochées au mur".

7

Générer des valeurs aléatoires : random

1

Affichez aléatoirement un entier entre 1 et 100 compris.

2

Affichez aléatoirement un fruit parmi une liste de cinq fruits.

3

Choisissez au hasard pile ou face("p" ou "f"). Demandez à l'utilisateur de faire son choix entre pile ou face. Si son choix est le même comme celui sélectionné au hasard par l'ordinateur, affichez le message "Vous avez gagné", sinon affichez "Pas de chance". Afficher un message qui donne le choix fait par l'ordinateur.

4

Choisissez au hasard un nombre entier entre 1 et 5(inclus). Demandez à l'utilisateur de choisir un entier naturel. Si l'utilisateur devine correctement, affichez le message "Bien joué", sinon, précisez si son choix est plus grand ou plus petit que la valeur cible. Si il devine correctement lors de sa deuxième supposition, affichez "Corriger", sinon affichez "Vous perdez".

5

Choisissez au hasard un nombre entier entre 1 et 10. Demandez à l'utilisateur d'entrer un entier naturel et de continuer à entrer des entiers naturels jusqu'à ce qu'il entre le nombre qui a été choisi au hasard.

6

Mettre à jour le programme 5 afin qu'il informe l'utilisateur si son choix est trop élevé ou trop bas avant qu'il choisisse à nouveau un autre nombre entier.

Faites un quiz de mathématiques qui pose cinq questions au hasard en générèrant deux nombres entiers pour faire la question(par exemple [num1] + [num2]). Demandez à l'utilisateur d'entrer la réponse. Si il réussit, ajoutez un point à son score. À la fin du quiz, dites combien il a de points sur cinq.

7

Affichez cinq couleurs et demandez à l'utilisateur d'en choisir une. Si il choisit la même couleur que le programme a choisie, affichez "C'est la bonne couleur", sinon affichez une réponse pleine d'esprit qui implique la bonne couleur, par exemple "Je parie que tu es ROUGE de honte" ou "Au-dessus des nuages, le ciel est toujours BLEU.". Tant que l'utilisateur n'a pas choisi la bonne couleur, continuez à lui donner le même indice et demandez à l'utilisateur d'entrer une couleur jusqu'à ce qu'il devine correctement.

8

n-uplets(tuples)| listes(lists) | dictionnaires(dictionaries)

1

Créez un tuple contenant les noms de cinq pays et affichez le tuple. Demandez à l'utilisateur d'entrer l'un des pays, puis affichez l'indice (c'est-à-dire la position dans la liste) de cet élément dans le tuple.

2

Modifiez le programme 1 pour demander à l'utilisateur d'entrer un entier naturel et affichez le pays à l'indice choisi.

3

Créez une liste de deux sports. Demandez à l'utilisateur quel est son sport préféré et ajoutez le à la fin de la liste. Triez la liste et affichez-la.

4

Créez une liste de six disciplines scolaires. Demandez à l'utilisateur quelle est la discipline de la liste qu'il préfère. Supprimez la discipline de la liste et affichez la liste.

5

Demandez à l'utilisateur d'entrer quatre de ses aliments préférés et les stocker dans un dictionnaire dont les éléments sont indicés à partir de 1. Affichez le dictionnaire sous la forme indice : élément. Demandez quel élément il faut supprimer et supprimez le du dictionnaire. Affichez le dictionnaire.

6

Entrez une liste de dix couleurs. Demandez à l'utilisateur un nombre entier naturel compris entre 0 et 4 et un second nombre entier naturel compris entre 5 et 9. Affichez la liste des couleurs dont l'indice est compris dans l'intervalle choisi par l'utilisateur.

7

Créez une liste de quatre entiers naturels à trois chiffres. Affichez la liste en affichant chaque élément sur une ligne séparée. Demandez à l'utilisateur d'entrer un entier naturel à trois chiffres. Si le nombre qu'il a donné est dans la liste, affichez l'indice de ce nombre, sinon, affichez le message "Il n'est pas dans la liste".

8

Demandez à l'utilisateur d'entrer les noms de trois personnes qu'il souhaite inviter à une fête et stockez-les dans une liste. Demandez à l'utilisateur si ensuite il souhaite ajouter un invité. Si c'est le cas, Permettez-lui d'ajouter des invités jusqu'à ce qu'il réponde « non ». Si il répond « non », affichez le nombre de personnes qu'il a invité à la fête.

Les listes 2D et dictionnaires

Créez ce qui suit à l'aide d'un liste 2D.

	0	1	2
0	2	5	8
1	3	7	4
2	1	6	9
3	4	2	0

1

À l'aide de la liste 2D du programme 1, demandez à l'utilisateur de sélectionner une ligne et une colonne et **affichez la valeur se trouvant à l'intersection de cette ligne et de cette colonne.**

2

En utilisant la liste 2D du programme 1, demandez à l'utilisateur la ligne qu'il souhaite afficher et **affichez la**. Demandez-lui d'entrer une nouvelle valeur et ajoutez-la à la fin de la ligne précédente et **affichez la ligne modifiée**.

3

Modifiez le programme 3 pour demander à l'utilisateur quelle ligne il veut afficher. **Affichez la**. Demandez quelle valeur de cette ligne il veut afficher et **affichez cette valeur**. Demandez à l'utilisateur s'il souhaite modifier la valeur. Si c'est le cas, Demandez la nouvelle valeur et modifiez la donnée. Enfin, **affichez de nouveau la ligne.**

4

Créez ce qui suit à l'aide d'un dictionnaire 2D montrant les ventes que chaque personne a réalisées dans les différentes régions géographiques

5

	N	S	E	W
John	3056	8463	8441	2694
Tom	4832	6786	4737	3612
Anne	5239	4802	5820	1859
Fiona	3904	3645	8821	2451

6

À l'aide du programme 5, demandez à l'utilisateur un nom et une région. Affichez les données dans un format convivial. Demandez à l'utilisateur le nom d'un vendeur et une région et donnez lui la possibilité de mettre à jour le montant des ventes. Affichez les ventes pour toutes les régions du vendeur dont le montant des ventes a été mis à jour.

7

Demandez à l'utilisateur d'entrer le nom, l'âge et la pointure pour quatre personnes. Demandez le nom de l'une des personnes de la liste et affichez son âge et sa pointure.

8

Adaptez le programme 7 pour qu'à la fin de la collecte des données s'affiche le nom et l'âge des quatre personnes.

9

Après avoir rassemblé les quatre noms, âges et pointures, demandez à l'utilisateur d'entrer le nom de la personne qu'il souhaite supprimer. Supprimez cette ligne des données et afficher les autres lignes sur des lignes séparées.

Les fichiers textes

1

Écrire un nouveau fichier appelé nombres.txt. Ajouter cinq chiffres au document qui sont stockés sur la même ligne et seulement séparés par un virgule. Inspecter alors le dossier qui contient votre script Python. Ouvrir le fichier texte obtenu dans le bloc-notes.

2

Créez un nouveau fichier appelé noms.txt. Ajoutez cinq noms au fichier, qui seront stockés sur des lignes séparées. Inspecter alors le dossier qui contient votre script python. Ouvrir le fichier texte obtenu dans le bloc-notes.

3

Ecrire un programme Python qui permet de lire les données du fichier noms.txt.

4

Ouvrez le fichier noms.txt. Demandez à l'utilisateur d'entrer un nouveau nom. Ajoutez celui-ci à la fin du fichier et affichez l'intégralité du fichier.

5

Affichez le menu suivant à l'utilisateur :

1) Créer un nouveau fichier.
2) Afficher les données du fichier.
3) Ajouter une donnée au fichier.
Sélectionner 1, 2 ou 3.

Demandez à l'utilisateur d'entrer 1, 2 ou 3. Si il fait un choix incorrecte affichez un message d'erreur. Si il sélectionne 1, créez le fichier sujet.txt(Si le fichier existe déjà il devra être écrasé). Si il sélectionne 2, affichez le contenu du fichier sujet.txt. Si il sélectionne 3, demandez à l'utilisateur de renseigner la donnée qu'il souhaite ajouter au fichier sujet.txt et ajoutez là en fin de fichier. Exécutez le programme un nombre de fois nécessaire pour vérifier son fonctionnement.

À l'aide du fichier noms.txt créé précédemment, affichez la liste des noms à l'aide d'un programme Python. Demandez à l'utilisateur de donner l'un des noms, puis enregistrez tous les noms sauf celui-ci dans un nouveau fichier nommé noms2.txt.

6

Les fonctions

1

Définissez une fonction qui renvoie un entier naturel demandé à l'utilisateur. Définissez une autre fonction qui utilise cet entier pour afficher les entiers naturels de 1 jusqu'à cet l'entier choisi par l'utilisateur.

2

Définir une fonction qui demandera à l'utilisateur une borne inférieure et une borne supérieure, puis retournez un entier naturel aléatoire entre ces deux bornes.

Définir une autre fonction qui :

1) Affiche l'instruction "Je pense à un entier naturel...",
2) demande à l'utilisateur de deviner la valeur de cet entier naturel.

Définir une troisième fonction qui :

Vérifiez si l'utilisateur a deviné la bonne valeur. Si oui, affichez le message "Correct, vous gagnez", sinon continuez à jouer en lui précisant si sa valeur est trop petite ou trop grande.

3

Affichez le menu suivant à l'utilisateur :

1) Addition
2) Soustraction

Entrez 1 ou 2

Si il entre 1, exécutez une fonction qui génère deux nombres entiers aléatoires entre 5 et 20, et qui demande à l'utilisateur de les additionner. La fonction renvoie un tuple avec la réponse de l'utilisateur et la bonne réponse.

Si il entre 2, exécutez une fonction qui génère un nombre entier aléatoire entre 25 et 50 et un autre nombre entier entre 1 et 25 et qui demande à l'utilisateur de soustraire le deuxième entier au premier. La fonction renvoie un tuple avec la réponse de l'utilisateur et la bonne réponse.

Créez une autre fonction qui vérifiera si la réponse est correcte. Si c'est le cas, affichez "Correct", sinon affichez "Incorrect, la réponse est » et affichez la bonne réponse.

Si l'utilisateur ne sélectionne pas une option possible dans le menu vous devez afficher un message approprié.

Créez un programme qui permettra à l'utilisateur de gérer facilement une liste de noms. Il faudra afficher un menu qui permettra à l'utilisateur d'ajouter un nom à la liste, de modifier un nom et de supprimer un nom de la liste ou afficher tous les noms de la liste. Il faudra ajouter une option au menu pour permettre à l'utilisateur de fermer le programme.

Si il sélectionne une option qui n'est pas dans le menu, alors il faudra afficher un message approprié. Après avoir sélectionner une option et réaliser l'opération associé, le menu doit de nouveau apparaitre pour réaliser une nouvelle opération sans avoir besoin de relancer le programme. Le programme doit être rendu aussi facile à utiliser que possible.

4

Affichez le menu suivant à l'utilisateur :

1) Ajouter une donnée au fichier.
2) Afficher les données.
3) Quitter le programme.

Entrez 1, 2 ou 3

Si l'utilisateur sélectionne 1, autorisez-le à ajouter à un fichier appelé salaires.csv le nom et le salaire d'un employé.

Si il sélectionne 2, affichez toutes les données du fichier salaires.csv.

Si il sélectionne 3, arrêtez le programme.

Si il sélectionne un option incorrecte affichez un message d'erreur.

Le programme doit continuer à tourner jusqu'à ce que l'option 3 soit choisie.

5

En Python, il n'est techniquement pas possible de supprimer un enregistrement d'un fichier csv. Au lieu de cela, vous avez besoin d'enregistrer les données du fichier dans une liste temporaire, de réaliser les modifications nécessaires et de remplacer le fichier d'origine avec les données de la liste temporaire.

Modifier le programme 5 pour proposer à l'utilisateur le menu suivant :

1) Ajouter une donnée au fichier.
2) Afficher les données.
3) Supprimer une donnée.
4) Quitter le programme.

Entrez 1, 2, 3 ou 4

6

Mémento v2.0.7

Mémento Python 3

Dernière version sur :
https://perso.limsi.fr/pointal/python:memento

Types de base

entier, flottant, booléen, chaîne, octets

```
int 783  0 -192  0b010 0o642 0xF3
          zéro   binaire octal hexa
float 9.23 0.0  -1.7e-6
                        ×10^-6
bool True  False
str "Un\nDeux"
     retour à la ligne échappé
    'L\'âme'
      ' échappé
```

Chaîne multiligne :
```
"""X\tY\tZ
1\t2\t3"""
```
tabulation échappée

```
bytes b"toto\xfe\775"
        hexadécimal  octal
```
immutables

Types conteneurs

- **séquences ordonnées**, accès par index rapide, valeurs répétables

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)  11,"y",7.4     ("mot",)  ()
```
Valeurs non modifiables (immutables) — *expression juste avec des virgules →* `tuple`

```
str bytes   "toto"   b"toto"
```
(séquences ordonnées de caractères / d'octets)

- **conteneurs clés**, sans ordre *a priori*, accès par clé rapide, chaque clé unique

```
dictionnaire dict {"clé":"valeur"}   dict(a=3,b=4,k="v")   {}
(couples clé/valeur)  {1:"un",3:"trois",2:"deux",3.14:"π"}
ensemble     set  {"clé1","clé2"}    {1,9,3,0}            set()
```
clés=valeurs hachables (types base, immutables…) — `frozenset` *ensemble immutable* — *vide*

Identificateurs

pour noms de variables, fonctions, modules, classes…

`a…zA…Z_` suivi de `a…zA…Z_0…9`

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☺ `a toto x7 y_max BigOne`
☹ ~~`0y and for`~~

Variables & affectation

`=`

affectation ⇔ **association** d'un *nom* à une *valeur*

1) évaluation de la valeur de l'expression de droite
2) affectation dans l'ordre avec les noms de gauche

```
x=1.2+8+sin(y)
a=b=c=0      affectation à la même valeur
y,z,r=9.2,-7.6,0   affectations multiples
a,b=b,a      échange de valeurs
a,*b=seq
*a,b=seq     dépaquetage de séquence en élément et liste
x+=3         incrémentation ⇔ x=x+3      et *= /= %= …
x-=2         décrémentation ⇔ x=x-2
x=None       valeur constante « non défini »
del x        suppression du nom x
```

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63
int(15.56) → 15
float("-11.24e8") → -1124000000.0
round(15.56,1) → 15.6
bool(x)  False pour x zéro, x conteneur vide, x None ou False ; True pour autres x
str(x) → "…"
chr(64) → '@'   ord('@') → 64
repr(x) → "…"
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"trois"),(1,"un")]) → {1:'un',3:'trois'}
set(["un","deux"]) → {'un','deux'}
```

spécification de la base du nombre entier en 2nd paramètre
troncature de la partie décimale
arrondi à 1 décimale (0 décimale → nb entier)
chaîne de représentation de x pour l'affichage (cf. Formatage au verso)
code ↔ caractère
chaîne de représentation littérale de x

str de jointure et séquence de str → str assemblée
```
':'.join(['toto','12','pswd']) → 'toto:12:pswd'
```
str découpée sur les blancs → list de str
```
"des mots  espacés".split() → ['des','mots','espacés']
```
str découpée sur str séparateur → list de str
```
"1,4,8,2".split(",") → ['1','4','8','2']
```
séquence d'un type → list d'un autre type (par liste en compréhension)
```
[int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Indexation conteneurs séquences

pour les listes, tuples, chaînes de caractères, bytes…

index négatif	−5	−4	−3	−2	−1	
index positif	0	1	2	3	4	
`lst=`	[10,	20,	30,	40,	50]	
tranche positive	0	1	2	3	4	5
tranche négative	−5	−4	−3	−2	−1	

Nombre d'éléments
`len(lst)→5`

index à partir de 0
(de 0 à 4 ici)

Accès individuel aux **éléments** par `lst[`*index*`]`
`lst[0]→10` *⇒ le premier* `lst[1]→20`
`lst[-1]→50` *⇒ le dernier* `lst[-2]→40`

Sur les séquences modifiables (`list`), suppression avec `del lst[3]` et modification par affectation `lst[4]=25`

Accès à des **sous-séquences** par `lst[`*tranche début*`:`*tranche fin*`:`*pas*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]   copie superficielle de la séquence
```

Indication de tranche manquante → à partir du début / jusqu'à la fin.
Sur les séquences modifiables (`list`), suppression avec `del lst[3:5]` et modification par affectation `lst[1:4]=[15,25]`

Logique booléenne

Comparateurs: `<` `>` `<=` `>=` `==` `!=`
(résultats booléens) ≤ ≥ = ≠

`a and b` et logique — *les deux en même temps*

`a or b` ou logique — *l'un ou l'autre ou les deux*

piège : `and` et `or` retournent la *valeur* de `a` ou de `b` *(selon l'évaluation au plus court).*
⇒ *s'assurer que* `a` *et* `b` *sont booléens.*

`not a` non logique

`True` `False` constantes Vrai/Faux

Blocs d'instructions

instruction parente :
bloc d'instructions 1…
instruction parente :
bloc d'instructions 2…
instruction suivante après bloc 1

indentation !

régler l'éditeur pour insérer 4 espaces à la place d'une tabulation d'indentation.

Imports modules/noms

module `truc` ⇔ *fichier* `truc.py`

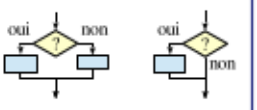
```
from monmod import nom1,nom2 as fct
```
→accès direct aux noms, renommage avec `as`
```
import monmod
```
→accès via `monmod.nom1` …

modules et packages cherchés dans le *python path* (cf. `sys.path`)

Instruction conditionnelle

un bloc d'instructions exécuté, uniquement ***si*** *sa condition est vraie*

`if` *condition logique* `:`
→ *bloc d'instructions*



Combinable avec des *sinon si, sinon si…* et un seul *sinon* final. Seul le bloc de la première condition trouvée vraie est exécuté.

```
if age<=18:
    etat="Enfant"
elif age>65:
    etat="Retraité"
else:
    etat="Actif"
```

avec une variable `x`:
`if bool(x)==True:` ⇔ `if x:`
`if bool(x)==False:` ⇔ `if not x:`

Maths

nombres flottants… valeurs approchées !

Opérateurs : `+` `-` `*` `/` `//` `%` `**`
Priorités `(…)` × ÷ ↑ ↑ a^b
÷ entière reste ÷

`@` → × matricielle *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```
priorités usuelles

angles en radians

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   ✓
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, etc.

Exceptions sur erreurs

Signalisation :
`raise` *ExcClass(…)*

Traitement :
```
try:
    → bloc traitement normal
except ExcClass as e:
    → bloc traitement erreur
```

bloc `finally` *pour traitements finaux dans tous les cas.*

Instruction boucle conditionnelle

*bloc d'instructions exécuté **tant que** la condition est vraie*

```
while condition logique :
    bloc d'instructions
```


```
s = 0          # initialisations avant la boucle
i = 1
while i <= 100:        # condition avec au moins une valeur variable (ici i)
    s = s + i**2
    i = i + 1          # faire varier la variable de condition !
print("somme:",s)
```

Algo : $s = \sum_{i=1}^{i=100} i^2$

⚠ attention aux boucles sans fin !

Contrôle de boucle

`break` sortie immédiate

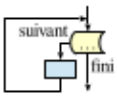
`continue` itération suivante

⚠ bloc `else` en sortie ***normale*** de boucle.

Instruction boucle itérative

*bloc d'instructions exécuté **pour chaque** élément d'un conteneur ou d'un itérateur*

```
for var in séquence :
    bloc d'instructions
```



Parcours des **valeurs** d'un conteneur

```
s = "Du texte"      # initialisations avant la boucle
cpt = 0
# variable de boucle, affectation gérée par l'instruction for
for c in s:
    if c == "e":
        cpt = cpt + 1
print("trouvé",cpt,"'e'")
```

Algo : comptage du nombre de e *dans la chaîne.*

boucle sur dict/set ⇔ boucle sur séquence des clés

utilisation des *tranches* pour parcourir un sous-ensemble d'une séquence

Parcours des **index** d'un conteneur séquence

- changement de l'élément à la position
- accès aux éléments autour de la position (avant/après)

```
lst = [11,18,9,12,23,4,17]
perdu = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        perdu.append(val)
        lst[idx] = 15
print("modif:",lst,"-modif:",perdu)
```

Algo: bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

⚠ bonne habitude : ne pas modifier la variable de boucle

Parcours simultané **index** et **valeurs** de la séquence :

```
for idx,val in enumerate(lst):
```

Affichage

```
print("v=",3,"cm :",x,",",y+4)
```

éléments à afficher : valeurs littérales, variables, expressions

Options de `print` :

- `sep=" "` séparateur d'éléments, défaut espace
- `end="\n"` fin d'affichage, défaut fin de ligne
- `file=sys.stdout` print vers fichier, défaut sortie standard

Saisie

```
s = input("Directives:")
```

⚠ `input` retourne toujours une **chaîne**, la convertir vers le type désiré (cf. encadré *Conversions* au recto).

Opérations génériques sur conteneurs

`len(c)` → nb d'éléments

`min(c)` `max(c)` `sum(c)`

`sorted(c)` → `list` *copie* triée

*Note: Pour dictionnaires et ensembles, ces opérations travaillent sur les **clés**.*

`val in c` → booléen, opérateur `in` de test de présence (`not in` d'absence)

`enumerate(c)` → *itérateur* sur (index, valeur)

`zip(c1,c2…)` → *itérateur* sur tuples contenant les éléments de même index des c_i

`all(c)` → `True` si **tout** élément de `c` évalué vrai, sinon `False`

`any(c)` → `True` si **au moins un** élément de `c` évalué vrai, sinon `False`

`c.clear()` supprime le contenu des dictionnaires, ensembles, listes

*Spécifique aux **conteneurs de séquences ordonnées** (listes, tuples, chaînes, bytes…)*

`reversed(c)` → *itérateur inversé* `c*5`→ duplication `c+c2`→ concaténation

`c.index(val)` → *position* `c.count(val)` → *nb d'occurences*

```
import copy
copy.copy(c)      → copie superficielle du conteneur
copy.deepcopy(c)  → copie en profondeur du conteneur
```

Opérations sur listes

⚠ modification de la liste originale

`lst.append(val)`	ajout d'un élément à la fin
`lst.extend(seq)`	ajout d'une séquence d'éléments à la fin
`lst.insert(idx,val)`	insertion d'un élément à une position
`lst.remove(val)`	suppression du premier élément de valeur *val*
`lst.pop([idx])`→*valeur*	supp. & retourne l'item d'index *idx* (défaut le dernier)
`lst.sort()` `lst.reverse()`	tri / inversion de la liste sur place

Opérations sur dictionnaires

`d[clé]=valeur` `del d[clé]`

`d[clé]` → *valeur*

`d.update(d2)` mise à jour/ajout des couples

`d.keys()` `d.values()` `d.items()` →*vues itérables sur les clés / valeurs / couples*

`d.pop(clé[,défaut])` → *valeur*

`d.popitem()` → *(clé,valeur)*

`d.get(clé[,défaut])` → *valeur*

`d.setdefault(clé[,défaut])` →*valeur*

Opérations sur ensembles

Opérateurs :

- `|` → union (caractère barre verticale)
- `&` → intersection
- `-` `^` → différence/diff. symétrique
- `<` `<=` `>` `>=` → relations d'inclusion

Les opérateurs existent aussi sous forme de méthodes.

`s.update(s2)` `s.copy()`

`s.add(clé)` `s.remove(clé)`

`s.discard(clé)` `s.pop()`

Fichiers

stockage de données sur disque, et relecture

```
f = open("fic.txt","w",encoding="utf8")
```

- **variable** fichier pour les opérations
- **nom** du fichier sur le disque (+chemin…) — cf modules `os`, `os.path` et `pathlib`
- **mode** d'ouverture
  - `'r'` lecture (read)
  - `'w'` écriture (write)
  - `'a'` ajout (append)
  - …`'+'` `'x'` `'b'` `'t'`
- **encodage** des caractères pour les *fichiers textes*: `utf8` `ascii` `latin1` …

en **écriture**

```
f.write("coucou")
f.writelines(list de lignes)
```

en **lecture**

⚠ *lit chaîne vide si fin de fichier*

`f.read([n])` → *caractères suivants* (*si n non spécifié, lit jusqu'à la fin !*)

`f.readlines([n])` → `list` *lignes suivantes*

`f.readline()` → *ligne suivante*

⚠ *par défaut mode texte* `t` *(lit/écrit* `str`*), mode binaire* `b` *possible (lit/écrit* `bytes`*). **Convertir de/vers le type désiré** !*

`f.close()` ⚠ ne pas oublier de **refermer le fichier** après son utilisation !

`f.flush()` écriture du cache `f.truncate([taille])` retaillage

lecture/écriture progressent séquentiellement dans le fichier, modifiable avec :

`f.tell()` →*position* `f.seek(position[,origine])`

Très courant : ouverture en bloc gardé (fermeture automatique) et boucle de lecture des lignes d'un fichier texte.

```
with open(…) as f:
    for ligne in f :
        # traitement de ligne
```

Séquences d'entiers

`range([début,] fin [,pas])`

⚠ *début* défaut 0, *fin* non compris dans la séquence, *pas* signé et défaut 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11

`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10

`range(len(séq))` → *séquence des index des valeurs dans séq*

⚠ *range fournit une séquence immutable d'entiers construits au besoin*

Définition de fonction

nom de la fonction (identificateur) — paramètres nommés

```
def fct(x,y,z):
    """documentation"""
    # bloc instructions, calcul de res, etc.
    return res
```

valeur résultat de l'appel, si pas de résultat calculé à retourner : `return None`

⚠ les paramètres et toutes les variables de ce bloc n'existent que *dans* le bloc et *pendant* l'appel à la fonction (penser "*boîte noire*")

Avancé : `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args nb variables d'arguments positionnels (→`tuple`), valeurs par défaut, **kwargs nb variable d'arguments nommés (→`dict`)*

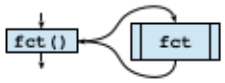
Appel de fonction

```
r = fct(3,i+2,2*i)
```

stockage/utilisation de la valeur de retour — *une valeur d'argument par paramètre*

⚠ c'est l'utilisation du nom de la fonction *avec les parenthèses* qui fait l'appel

*Avancé: *séquence **dict*



Opérations sur chaînes

`s.startswith(prefix[,début[,fin]])` `s.endswith(suffix[,début[,fin]])` `s.strip([caractères])`

`s.count(sub[,début[,fin]])` `s.partition(sep)` → *(avant,sep,après)*

`s.index(sub[,début[,fin]])` `s.find(sub[,début[,fin]])`

`s.is…()` *tests sur les catégories de caractères (ex. `s.isalpha()`)*

`s.upper()` `s.lower()` `s.title()` `s.swapcase()`

`s.casefold()` `s.capitalize()` `s.center([larg,rempl])`

`s.ljust([larg,rempl])` `s.rjust([larg,rempl])` `s.zfill([larg])`

`s.encode(codage)` `s.split([sep])` `s.join(séq)`

Formatage

directives de formatage — valeurs à formater

```
"modele{} {} {}".format(x,y,r)  → str
```

`"{`*sélection* `:` *formatage* `!` *conversion*`}"`

- **Sélection** :
  - `2`
  - `nom`
  - `0.nom`
  - `4[clé]`
  - `0[2]`
- **Formatage** :

car-rempl. *alignement* *signe* *larg.mini* . *précision~larg.max* *type*

`< > ^ =` `+ - espace` `0` au début pour remplissage avec des 0

entiers : `b` binaire, `c` caractère, `d` décimal (défaut), `o` octal, `x` ou `X` hexa…

flottant : `e` ou `E` exponentielle, `f` ou `F` point fixe, `g` ou `G` approprié (défaut),

chaîne : `s` … `%` pourcentage

- **Conversion** : `s` (texte lisible) ou `r` (représentation littérale)

Exemples

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="L'ame")
→'"L\'ame"'
```