Défi 1 Échanger le matériel

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Sauvegarder une valeur avant de la remplacer.

1. Imaginer deux boîtes A et B contenant deux objets différents. Pour les échanger, une troisième boîte peut être utile.

2. Copier d'abord une valeur dans une variable provisoire. Une affectation remplace l'ancienne valeur de la variable à gauche du signe =.

3. Après cette sauvegarde, réaliser l'échange en utilisant uniquement les noms des variables.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
casier_a = "Calculatrice"
casier_b = "Casque"
print("Avant :", casier_a, casier_b)

temporaire = ___
casier_a = ___
casier_b = ___

print("Après :", casier_a, casier_b)
```

Vérifier avant de demander une autre aide

Remplacer seulement les valeurs de départ par "Tablette" et "Câble". Le même échange doit fonctionner.

Si casier_a est remplacé trop tôt, où retrouver son ancienne valeur ?

.....

Défi 2 Construire un identifiant de casier

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Construire un texte puis en extraire une partie.

1. Conserver les deux saisies comme textes, pour garder les zéros du numéro.

2. Assembler trois éléments dans cet ordre : bâtiment, tiret, numéro. Pour des textes, + réalise cet assemblage.

3. La longueur se mesure avec len. Pour l'extraction, utiliser identifiant et la notation [-3:].

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
batiment = input("Bâtiment : ")
numero = input("Numéro sur trois caractères : ")

identifiant = ___
print("Identifiant :", identifiant)
print("Longueur :", ___)
print("Numéro extrait :", ___)
```

Vérifier avant de demander une autre aide

Tester B / 007 puis SIN / 012. Le tiret compte comme un caractère. Le numéro extrait doit toujours garder ses trois caractères.

Pourquoi int(numero) empêcherait-il de conserver correctement 007 ?

.....

Défi 3 Respecter les horaires

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Combiner un bon code et une plage horaire.

1. Traiter d'abord le mauvais code. Dans ce cas, le résultat ne dépend pas de l'heure.

2. Pour un bon code, vérifier deux limites à la fois : l'heure minimale est incluse et l'heure maximale est exclue.

3. Utiliser and pour demander que les deux comparaisons soient vraies.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
code_saisi = input("Code : ")
heure = int(input("Heure de 0 à 23 : "))

if ___:
    print("ACCÈS REFUSÉ")
else:
    if ___:
        print("OUVERTURE AUTORISÉE")
    else:
        print("HORS HORAIRES")
```

Vérifier avant de demander une autre aide

Tester un bon code à 8 h puis à 18 h. Tester aussi un mauvais code à une heure autorisée et à une heure interdite.

Avec un code faux à 20 h, quel contrôle doit déterminer le message ?

.....

Défi 4 Détecter la perte du zéro

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Comparer la saisie originale et sa conversion.

1. Observer ce que deviennent les zéros à gauche lorsque int transforme le texte en nombre.
2. La règle exige un texte exact. Deux textes différents peuvent-ils donner le même entier ?
3. Après les tests, réparer uniquement la condition : comparer les textes en conservant le zéro initial du code attendu.

Niveau 2 Compléter une structure

Le code ci-dessous contient volontairement le bug à étudier. Le tester, puis le réparer.

```
code_saisi = input("Code : ")

# Condition volontairement erronée à étudier.
if int(code_saisi) == 427:
    print("AUTORISÉ")
else:
    print("REFUSÉ")
```

Vérifier avant de demander une autre aide

Ce programme fonctionne, mais certaines autorisations sont incorrectes. Tester 0427, 427, 00427 et 9999 ; comparer chaque résultat à la règle avant de corriger.

Quels codes différents deviennent le même nombre après la conversion ?

.....

Défi 5 Limiter à trois tentatives

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Distinguer le nombre de saisies et la réussite.

1. Créer un compteur avant la boucle. L'augmenter à chaque saisie, correcte ou incorrecte.
2. Continuer seulement si le code reste faux ET s'il reste une tentative.
3. Après la boucle, vérifier le code pour distinguer réussite et blocage : le troisième essai peut réussir.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
code_attendu = "0427"
code_saisi = ""
tentatives = 0

while ___:
    code_saisi = input("Code : ")
    tentatives = ___

if ___:
    print("OUVERTURE AUTORISÉE")
else:
    print("CASIER BLOQUÉ")
print("Tentatives utilisées :", tentatives)
```

Vérifier avant de demander une autre aide

La chaîne vide initiale permet d'entrer dans la boucle. Vérifier les trois scénarios de la fiche ; aucun ne doit demander une quatrième saisie.

Pourquoi tentatives == 3 ne permet-il pas, à lui seul, de conclure au blocage ?

.....

Défi 6 Compter les codes refusés

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Compter chaque tentative refusée dans toute la liste.

1. Avant la boucle, créer un compteur à 0 : aucun code n'a encore été analysé.

2. Dans la boucle, ajouter 1 au compteur si le code est différent de "0427". Un bon code ne modifie pas le total. Deux mauvais codes identiques comptent comme deux refus.

3. Après la boucle, afficher le total une seule fois. != signifie « différent de ».

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
essais = ["1111", "0427", "427", "9999", "0427"]
refus = ___

for code_saisi in essais:
    if code_saisi != ___:
        refus = ___

print("Nombre de refus :", ___)
```

Vérifier avant de demander une autre aide

La liste fournie doit donner 3 refus. Tester ["0427", "0427"] → 0 ; ["1111", "1111", "9999"] → 3 ; [] → 0. Si plusieurs totaux apparaissent, vérifier que print est placé après la boucle.

Pourquoi faut-il conserver le compteur lorsqu'un code correct apparaît ?

.....

Défi 7 Utiliser plusieurs codes attendus

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Transmettre le code attendu entre les fonctions.

1. verifier_acces reçoit deux textes et les compare : aucun code secret ne doit être fixé dans son corps.
2. compter_refus doit transmettre le code saisi ET le code attendu lorsqu'elle appelle verifier_acces.
3. Compter les résultats "REFUSÉ", puis renvoyer le total après la boucle.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
def verifier_acces(code_saisi, code_attendu):
    if ___:
        return "AUTORISÉ"
    else:
        return "REFUSÉ"

def compter_refus(essais, attendu):
    compteur = 0
    for code_saisi in essais:
        resultat = ___
        if ___:
            compteur = ___
    return ___

codes = ["0427", "0427", "1357"]
print(compter_refus(codes, "0427"))
print(compter_refus(codes, "1357"))
```

Vérifier avant de demander une autre aide

Ajouter un appel avec une liste vide. Changer le code attendu uniquement dans l'appel : les définitions des fonctions doivent rester identiques.

Quelle valeur correspond au paramètre code_attendu lors de l'appel depuis compter_refus ?

.....

Défi 8 Maintenir le blocage

COUP DE POUCE • À utiliser avec la fiche mission clarifiée

Commencer par le niveau 1. Masquer la partie code avec une feuille et ne la découvrir qu'en cas de besoin.

Niveau 1 Chercher une méthode

Mémoriser le blocage indépendamment du code saisi.

1. Utiliser un compteur d'échecs successifs et une variable bloque. False signifie non bloqué ; True signifie bloqué.

2. Vérifier bloque avant d'appeler verifier_code. Une fois bloqué, afficher seulement BLOQUÉ pour tous les essais suivants.

3. Avant le blocage, une réussite remet le compteur à zéro. Un échec l'augmente. Au troisième échec, changer bloque et afficher BLOQUÉ.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Effectuer aussi les tâches indiquées dans les commentaires avant d'exécuter.

```
# Copier ici la définition de verifier_code.
essais = ["1111", "2222", "3333", "0427"]
echecs = 0
bloque = False

for code_saisi in essais:
    if bloque:
        print("BLOQUÉ")
    else:
        resultat = ___
        if ___:
            echecs = ___
        else:
            echecs = ___
        if ___:
            bloque = ___
            print("BLOQUÉ")
        else:
            print(resultat)
```

Vérifier avant de demander une autre aide

Après trois échecs, le bon code ne doit plus ouvrir. Vérifier aussi une réussite entre deux séries d'échecs : elle doit remettre le compteur à zéro avant tout blocage.

Quelle branche empêche de rappeler verifier_code après le blocage ?

.....