Défi 1 Prévoir une récolte

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

Relier les quatre données aux trois résultats.

1. Commencer par le nombre de plants : combien de plants obtient-on en réunissant tous les bacs ?
2. Chaque plant produit plusieurs portions. Utiliser le résultat précédent pour calculer la récolte totale.
3. Une portion est consommée par repas. Quelle opération permet de calculer ce qui reste ?

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
bacs = 4
plants_par_bac = 6
portions_par_plant = 3
repas = 50

plants = ___
portions = ___
reste = ___

print("Plants :", ___)
print("Portions produites :", ___)
print("Portions restantes :", ___)
```

Vérifier avant de demander une autre aide

Remplacer seulement bacs par 5. Les trois résultats doivent changer sans modifier les print. Comparer aux tests de la fiche défi.

Quelle variable faut-il utiliser pour éviter d'écrire directement 50 dans le calcul du reste ?

.....

Défi 2 Calculer le nombre d'arrosages

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 les arrosages complets et l'eau restante.

1. Dessiner une réserve de 17 L et retirer des groupes de 5 L : combien de groupes complets peut-on retirer ? Combien de litres restent ?

2. // donne le nombre de groupes complets ; % donne le reste. Choisir les deux nombres et leur ordre.

3. Les saisies sont des entiers. La consommation est strictement positive.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
reserve = int(input("Réserve en litres : "))
consommation = int(input("Litres par arrosage : "))

arrosages = ___
reste = ___

print("Arrosages complets :", arrosages)
print("Litres restants :", reste)
```

Vérifier avant de demander une autre aide

Vérifier que arrosages × consommation + reste retrouve la réserve. Le reste doit être inférieur à la consommation.

Pourquoi le calcul avec / ne donne-t-il pas directement le nombre d'arrosages complets ?

.....

Défi 3 Arroser seulement si possible

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

Choisir quelle condition vérifier en premier.

1. Avec une réserve trop faible, l'arrosage est interdit, quelle que soit l'humidité. Traiter ce cas en premier.
2. Si la réserve suffit, il reste deux possibilités : terre sèche ou humidité suffisante.
3. Écrire un seul affichage dans chacune des trois situations.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
humidite = int(input("Humidité en % : "))
reserve = int(input("Réserve en litres : "))

if ___:
    print("RÉSERVE FAIBLE")
else:
    if ___:
        print("ARROSER")
    else:
        print("ATTENDRE")
```

Vérifier avant de demander une autre aide

Tester les valeurs exactes 10 L et 35 %. Relire les mots « inférieure à » pour choisir entre < et <=.

Pourquoi la vérification de l'humidité est-elle placée dans le else ?

.....

Défi 4 Trouver un bug au seuil

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

Trouver un test qui révèle la mauvaise condition.

1. Comparer les questions « est-ce strictement plus grand ? » et « est-ce plus grand ou égal ? ». Dans quel cas leurs réponses diffèrent-elles ?

2. Les températures 20 et 30 ne suffisent pas. Chercher une valeur au bord de la règle.

3. Noter séparément ce que la règle demande et ce que le programme fait. Réparer ensuite.

Niveau 2 Compléter une structure

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

```
temperature = float(input("Température : "))

# Condition volontairement erronée à étudier.
if temperature >= 28:
    print("ACTIVE")
else:
    print("ARRÊTÉE")
```

Vérifier avant de demander une autre aide

Ce programme est exécutable mais sa décision peut être fausse. Tester 20, 30 et la valeur choisie, puis modifier uniquement l'opérateur fautif.

Mon troisième test : ..... ; message attendu : ..... ; message observé avant réparation : .....

.....

Défi 5 Stabiliser la ventilation

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

Conserver une décision entre deux mesures.

1. Une variable etat doit exister avant le while. Elle garde sa valeur d'un tour au suivant.
2. Changer etat seulement lorsque la mesure franchit le seuil de démarrage ou celui d'arrêt.
3. Entre les deux seuils, ne rien affecter à etat. Afficher son contenu après les conditions.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
etat = "ARRÊTÉE"
temperature = float(input("Température ou -99 : "))

while temperature != -99:
    if ___:
        etat = ___
    elif ___:
        etat = ___
    print(etat)
    temperature = float(input("Température ou -99 : "))

print("Surveillance terminée")
```

Vérifier avant de demander une autre aide

Suivre à la main les saisies 29 puis 27. La seconde mesure est-elle une raison d'arrêter selon la règle du défi ? Tester aussi -99 dès le début.

Pourquoi ne faut-il pas remettre etat à "ARRÊTÉE" au début de chaque tour ?

.....

Défi 6 Mesurer une période chaude

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 la série en cours et la meilleure série.

1. Un premier compteur suit la longueur de la série chaude en cours. Une mesure non chaude coupe cette série.

2. Un deuxième compteur conserve la plus grande longueur rencontrée. Il ne doit pas perdre le record quand la série s'arrête.

3. Comparer la longueur en cours au record chaque fois qu'une nouvelle mesure chaude allonge la série.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
mesures = [26, 29, 30, 27, 31, 32, 33, 28]
serie = 0
maximum = 0

for temperature in mesures:
    if ___:
        serie = ___
        if ___:
            maximum = ___
    else:
        serie = ___

print("Plus longue série :", maximum)
```

Vérifier avant de demander une autre aide

Tester une liste qui se termine par des mesures chaudes et une liste vide. Il ne faut pas attendre une mesure froide pour enregistrer le record.

Après [29, 30, 27], quelles valeurs doivent avoir serie et maximum ?

.....

Défi 7 Compter avec une fonction

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 les décisions renvoyées par une fonction.

1. Reprendre choisir_ventilation de la mission 7. Lui transmettre chaque mesure, puis récupérer le texte qu'elle renvoie.

2. Augmenter le compteur uniquement lorsque ce texte vaut "ACTIVE".

3. Renvoyer le compteur quand toutes les mesures ont été traitées. Le print appartient au programme principal.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
# Copier ici la définition de choisir_ventilation.

def compter_activations(mesures):
    compteur = 0
    for temperature in mesures:
        etat = ___
        if ___:
            compteur = ___
    return ___

print(compter_activations([22, 29, 28, 31]))
print(compter_activations([]))
```

Vérifier avant de demander une autre aide

La condition temperature > 28 doit rester seulement dans choisir_ventilation. Pour une liste vide, le compteur doit pouvoir être renvoyé sans entrer dans le for.

À quel niveau d'indentation faut-il placer return pour traiter toute la liste ?

.....

Défi 8 Repérer un capteur défaillant

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

Contrôler la mesure avant de commander la ventilation.

1. Une température hors de la plage 0 à 50 est suspecte. La vérifier avant le seuil de ventilation.
2. Le résultat de la fonction doit être l'un des trois textes prévus.
3. Dans le programme principal, utiliser deux compteurs. Classer chaque résultat dans une seule catégorie.

Niveau 2 Compléter une structure

Remplacer chaque ___ par le code nécessaire. Compléter aussi les commentaires de travail avant d'exécuter.

```
def choisir_ventilation(temperature):
    if ___:
        return "CAPTEUR À VÉRIFIER"
    elif ___:
        return "ACTIVE"
    else:
        return "ARRÊTÉE"

mesures = [-2, 0, 28, 29, 50, 51]
anomalies = 0
activations = 0
for temperature in mesures:
    etat = ___
    print(temperature, etat)
    if ___:
        anomalies = ___
    elif ___:
        activations = ___

# Ajouter les deux affichages des totaux.
```

Vérifier avant de demander une autre aide

Une condition « hors plage » combine deux cas avec or. Vérifier 0, 50 et 50.1. Une mesure suspecte ne doit jamais augmenter activations.

Pourquoi faut-il vérifier la plage avant la condition de ventilation ?

.....