Corrigé détaillé des modules 5 et 6

BTS SIO1 • Deux séances de 60 minutes • Document enseignant

La fiche 5 consolide les structures de contrôle et distingue compteur, cumul et seuil. La fiche 6 introduit les fonctions avec des entiers, puis leur assemblage dans un défi. Les situations reprennent le travail sur les stocks ; les calculs restent accessibles sans formule mathématique.

Distribuer et accompagner

Fiche 5 : 4 pages, dont un défi facultatif en page 4. Fiche 6 : 5 pages, dont le défi en page 4 et son aide complète en page 5. Prévoir un éditeur Python 3 si disponible ; les traces et prévisions sont possibles sur papier. Les productions longues se font au dos ou dans l'éditeur. Conserver les aides accessibles après un premier essai.

Les défis sont proposés aux élèves qui ont terminé le parcours, ou repris à la séance suivante. Ils ne s'ajoutent pas aux 60 minutes obligatoires. La fiche 6 ne nécessite ni listes, ni input(), ni récursivité. Le type entier est explicité dans les signatures en pseudo-code.

Déroulés minutés

Étape	Fiche 5	Fiche 6
Choisir / comprendre un appel	0 à 8 min	0 à 12 min
Comprendre les traces	8 à 20 min	Inclus dans la première étape
Compléter	20 à 28 min	12 à 22 min
Traduire	28 à 38 min	22 à 32 min
Déboguer	38 à 46 min	32 à 42 min
Créer	46 à 57 min	42 à 57 min
Bilan individuel	57 à 60 min	57 à 60 min

Réussites à observer

Fiche 5 : l'élève justifie son choix de structure, met à jour deux variables sans les confondre, teste l'égalité au seuil et prévoit un cas sans passage. Fiche 6 : il distingue définition et appel, paramètre et argument, return et print ; il réutilise un résultat et teste plusieurs arguments.

Utiliser les aides

Après l'indice, laisser le temps de reprendre. Après le pseudo-code complet, demander une explication de ligne et une nouvelle prévision. Une erreur de syntaxe avec une trace juste n'appelle pas la même reprise qu'un programme qui s'exécute mais calcule une mauvaise valeur.

Préparer progressivement le CCF

Les défis « Vers le CCF » entraînent la lecture d'un problème, la complétion du pseudo-code, l'explication d'un rôle, la traduction et les essais. Le remplissage progressif s'inspire du mécanisme de cumul et de seuil du sujet de tirelire fourni. Ces exercices courts ne constituent pas une épreuve blanche complète.

Si le temps manque, corriger le débogage collectivement et reprendre la création au cours suivant. La définition et l'appel d'une fonction doivent être compris avant d'exiger l'assemblage des deux fonctions du défi.

Module 5 Corrigé des activités

Choisir une structure

1. if : on décide d'afficher une alerte selon un état. 2. for : quatre postes fixent quatre passages. 3. while : on reteste si le stock est inférieur à 8. Une autre structure peut produire un résultat correct ; demander pourquoi elle serait pertinente, puis revenir à la structure la plus directe ici.

Comprendre les traces

Boucle A	Départ	Passage 1	Passage 2	Passage 3
total	5	7	9	11

A affiche 11 une seule fois. B : stock passe de 3 à 5 puis à 7 ; ajouts passe de 0 à 1 puis à 2. Affichages : 7 puis 2. La condition est testée trois fois pour deux passages. Si stock vaut 7 dès le départ, aucun passage : stock reste 7 et ajouts vaut 0.

Compléter et traduire

Trous : 0 ; 4 ; total + 3. Résultat : 12. total mémorise le cumul ; jour repère les passages.

```
total = 0
for jour in range(1, 5):
    total = total + 3
print(total)
```

Déboguer le seuil

Le programme initial affiche 10 puis 5. Il réalise cinq livraisons et s'arrête à l'égalité, alors que l'objectif exige plus de 10. Correction : while total <= 10:. Le programme corrigé affiche 12 puis 6. À total = 10, la condition doit donc rester vraie.

Créer une synthèse

```
Pseudo-code
stock ← 2
Pour livraison de 1 à 3 Faire
    stock ← stock + 2
Fin Pour
Si stock >= 8 Alors
    Afficher "Objectif atteint"
Sinon
    Afficher "À compléter"
Fin Si
Afficher stock

Python
stock = 2
for livraison in range(1, 4):
    stock = stock + 2
if stock >= 8:
    print("Objectif atteint")
else:
    print("À compléter")
print(stock)
```

Départ 2 : stock final 8, Objectif atteint. Départ 0 : stock final 6, À compléter. Le if intervient après les trois livraisons. Accepter un affichage du stock avant le message si l'intention est respectée. Bilan : compteur augmenté de 1 ; continuer tant que total <= 10.

Module 5 Corrigé du défi

Variables et pseudo-code

total est le cumul reçu ; jours compte les jours déjà traités ; lot est la prochaine quantité à ajouter au début de chaque passage. Trous : total <= 14 ; total + lot ; jours + 1 ; lot + 1.

```
total ← 0
jours ← 0
lot ← 2
Tant Que total <= 14 Faire
    total ← total + lot
    jours ← jours + 1
    lot ← lot + 1
Fin Tant Que
Afficher jours
Afficher total
```

Traduction en Python

```
total = 0
jours = 0
lot = 2
while total <= 14:
    total = total + lot
    jours = jours + 1
    lot = lot + 1
print(jours)
print(total)
```

Fin du jour	Lot ajouté	Total	Prochain lot
1	2	2	3
2	3	5	4
3	4	9	5
4	5	14	6
5	6	20	7

Le programme affiche 5 puis 20. À la fin du jour 4, total vaut 14 : l'objectif strict n'est pas encore atteint. Le jour 5 est nécessaire. Pour un seuil de 13, la première valeur qui le dépasse est 14 au jour 4.

Relances et erreurs à anticiper

Si lot est augmenté avant l'ajout, le premier apport devient 3 : faire tracer seulement le premier jour. Si jours est initialisé à 1, il compte un jour de trop. Si total est remis à 0 dans la boucle, le cumul est perdu. Si la condition est total < 14, le programme s'arrête à 14 au jour 4.

Expliquer l'arrêt sur ces données : chaque lot est positif, le total augmente à chaque passage et finit par dépasser le seuil fini. Le prochain lot vaut 7 à la sortie, mais le dernier lot ajouté vaut 6 : ne pas les confondre.

Module 6 Corrigé des premiers appels

Comprendre

Expression ou variable	Valeur
apres_retour(0)	2
apres_retour(5)	7
arrivee	5
depart	3

Affichages : 5 puis 3. Paramètre : stock. Type du résultat : entier. Lors de apres_retour(depart), l'argument a la valeur 3. Le calcul stock + 2 retourne 5, qui est affecté à arrivee ; depart reste 3.

```
def apres_retour(stock):
    return stock + 2

depart = 3
arrivee = apres_retour(depart)
print(arrivee)
print(depart)
```

La ligne def définit la fonction sans effectuer le calcul. Le corps s'exécute lors de l'appel. Rester sur ces exemples d'entiers pour expliquer le paramètre local ; ne pas généraliser aux effets de modification de listes, qui seront étudiés plus tard.

Compléter et traduire

Trous : stock - 1 ; restant. apres_pret(6) retourne 5 ; apres_pret(1) retourne 0. La fonction n'affiche rien : seul le programme principal utilise Afficher / print.

```
Pseudo-code
Fonction apres_pret(stock : entier) : entier
    Retourner stock - 1
Fin Fonction
restant ← apres_pret(6)
Afficher restant

Python
def apres_pret(stock):
    return stock - 1

restant = apres_pret(6)
print(restant)
```

Déboguer

Dans le code fautif, le print interne affiche 5, mais ne retourne pas ce résultat. L'appel se termine sans return : Python renvoie None, affecté à restant puis affiché. Remplacer uniquement print(stock - 1) par return stock - 1. Le code corrigé est celui ci-dessus et affiche 5 une seule fois.

Vérifier la compréhension

Demander : « Où la valeur 6 entre-t-elle ? Où la valeur 5 sort-elle ? Où est-elle mémorisée ? ». Si l'élève écrit apres_pret sans parenthèses, faire distinguer le nom de la fonction de son appel apres_pret(6).

Module 6 Créer une fonction et cumuler

Corrigé de la création

```
Fonction besoin(stock : entier) : entier
    Si stock < 5 Alors
        Retourner 5 - stock
    Sinon
        Retourner 0
    Fin Si
Fin Fonction

Python
def besoin(stock):
    if stock < 5:
        return 5 - stock
    else:
        return 0

print(besoin(2))
```

Stock	Résultat retourné	Explication
2	3	Il manque 3 casques pour atteindre 5.
5	0	Le seuil est atteint.
7	0	Aucun casque ne manque.

Les deux branches retournent un entier. La fonction n'affiche rien ; print(besoin(2)) affiche 3 dans le programme principal. Accepter un return 0 après le if, sans else, si l'élève comprend que le premier return termine l'appel lorsque stock < 5.

Bilan : def ; return ; besoin(2). Ne pas confondre le type annoncé dans le pseudo-code avec le nom de la fonction ou de son paramètre.

Défi Compléter les deux fonctions

Trous : jour + 1 ; n ; lot(jour) ; total. contenu reçoit un nombre de jours et retourne le cumul. La boucle appelle lot avec le jour courant, de 1 à n.

```
Fonction lot(jour : entier) : entier
    Retourner jour + 1
Fin Fonction

Fonction contenu(n : entier) : entier
    total ← 0
    Pour jour de 1 à n Faire
        total ← total + lot(jour)
    Fin Pour
    Retourner total
Fin Fonction
```

```
def lot(jour):
    return jour + 1

def contenu(n):
    total = 0
    for jour in range(1, n + 1):
        total = total + lot(jour)
    return total
```

Module 6 Assembler et tester le défi

lot(n) répéterait le lot du dernier jour à chaque passage au lieu d'additionner des lots différents. Exemple pour n = 2 : 3 + 3 donnerait 6 au lieu de 2 + 3 = 5. return total est placé après la boucle, sinon contenu s'arrêterait au premier passage.

Programme principal

Trous : contenu(jours) <= 14 ; jours + 1 ; contenu(jours). Les deux fonctions doivent être définies avant ce programme principal.

```
jours ← 0
Tant Que contenu(jours) <= 14 Faire
    jours ← jours + 1
Fin Tant Que
Afficher jours
Afficher contenu(jours)
```

Programme Python complet

```
def lot(jour):
    return jour + 1

def contenu(n):
    total = 0
    for jour in range(1, n + 1):
        total = total + lot(jour)
    return total

jours = 0
while contenu(jours) <= 14:
    jours = jours + 1
print(jours)
print(contenu(jours))
```

Test	Résultat attendu
lot(1)	2
lot(4)	5
contenu(0)	0
contenu(2)	5
contenu(4)	14
Premier jour au-delà de 14	5
Contenu à ce jour	20

Pour contenu(0), range(1, 1) ne fournit aucune valeur : total reste 0 et return renvoie 0. À chaque appel, contenu réinitialise son propre total et recalcule le cumul. Le programme principal teste successivement les contenus 0, 2, 5, 9, 14, 20.

Conduire la correction

Tester d'abord lot, puis contenu, puis l'assemblage. Si contenu(2) donne 2, vérifier la place du return. Si contenu(2) donne 6, vérifier l'argument de lot. Si le programme s'arrête au jour 4, revenir à l'égalité au seuil. Pour vérifier l'autonomie après l'aide, demander le résultat avec un seuil de 13 : jour 4, contenu 14.

Cette version privilégie la décomposition en fonctions ; elle recalcule les sommes, ce qui reste acceptable pour les petits exemples. La fiche 5 montre une version qui conserve le cumul. Aucune optimisation ni récursivité n'est attendue ici.