

Correction DS bases de la programmation

Durée de l'épreuve : **01h50**

L'usage de la calculatrice n'est pas autorisé.

Le candidat répond sur feuilles doubles numérotées et garde l'énoncé.

La qualité de la rédaction, la clarté et la précision seront prises en compte.

Les traces de recherche, même incomplètes ou infructueuses, seront valorisées.

Exercice 1 (9 points)

1. Types et valeurs de variables (3 pt)

Donner le type des variables finales (**res_1**, **res_2** et **res_3**) et déterminer leur valeur après exécution des programmes ci-dessous :

```
# programme 1
a = 17
b = 4
res_1 = a / b + a // b
```

```
# programme 2
chaine = "15"
n = 3
res_2 = chaine * n
```

```
# programme 3
x = 8
y = 12
res_3 = (x > 5) and not (y < 10)
```

```
print(f"Le type de la variable res_1 est {type(res_1)} et sa valeur est {res_1}")
print(f"Le type de la variable res_2 est {type(res_2)} et sa valeur est {res_2}")
print(f"Le type de la variable res_3 est {type(res_3)} et sa valeur est {res_3}")

"""
Le type de la variable res_1 est <class 'float'> et sa valeur est 8.25
Le type de la variable res_2 est <class 'str'> et sa valeur est 151515
Le type de la variable res_3 est <class 'bool'> et sa valeur est True
"""
```

2. Suivi de variables (3 pt)

Écrire ce qui s'affichera en console lors de l'exécution des programmes ci-dessous :

```
# programme 1
a = 7
b = 3
c = a + b
print(a, b, c)
a = a * b
print(a, b, c)
b = c - 2
print(a, b, c)
```

```
7 3 10
21 3 10
21 8 10
```

```
# programme 2
x = 2
y = 0
for i in range(3):
    y = y + x
    x = x + 1
    print(i, x, y)
```

```
0 3 2
1 4 5
2 5 9
```

```
# programme 3
val_a = 1
val_b = 10
compteur = 1
while val_b > val_a:
    val_a = val_a + 2
    val_b = val_b - 1
    print(compteur, val_a, val_b, val_b > val_a)
    compteur = compteur + 1
```

```
1 3 9 True
2 5 8 True
3 7 7 False
```

3. Débogage (3 pt)

Identifiez les erreurs des programmes ci-dessous et les corriger :

```
# programme 1
age = input("Entrez votre âge : ")
annees_restantes = 18 - age
print(f"Plus que {annees_restantes} ans avant la majorité.")
```

```
# on convertit en int
age = int(input("Entrez votre âge : "))
# sinon : TypeError: unsupported operand type for -: 'int' and 'str'
annees_restantes = 18 - age
print(f"Plus que {annees_restantes} ans avant la majorité.")
```

```
# programme 2
note1 = 14
note2 = "16"
moyenne = (note1 + Note2) / 2
print("La moyenne est " + moyenne)
```

```
note1 = 14
note2 = 16
# TypeError: unsupported operand type for +: 'int' and 'str'
moyenne = (note1 + note2) / 2 # NameError: name 'Note2' is not defined
# unsupported operand type for +: 'str' and 'float'
print("La moyenne est " + str(moyenne))
```

```
# programme 3
def calculer_ttc(prix_ht)
    prix_ttc = prix_ht * 1.20
    print(prix_ttc)

calculer_ttc(100)
print(f"Le prix final est de {prix_ttc} euros.")
```

```
def calculer_ttc(prix_ht):      # manque les deux points (SyntaxError)
    prix_ttc = prix_ht * 1.20
    return prix_ttc           # pas de return

# Après avoir corrigé le renvoi, on affecte la variable prix_ttc :
prix_ttc = calculer_ttc(100)
# sinon : NameError: name 'prix_ttc' is not defined
print(f"Le prix final est de {prix_ttc} euros.")
```

Exercice 2 (6 points)

Une entreprise de logistique calcule le **tarif de livraison** d'un colis en fonction de son poids (un entier exprimé en grammes). Les règles de tarification de base sont les suivantes :

- Tout poids inférieur ou égal à **250 g** est facturé **4 €**.
- Tout poids supérieur à **250 g** et inférieur ou égal à **1 kg** (1000 g) est facturé **7 €**.
- Tout poids strictement supérieur à **1 kg** est facturé **12 €**.

1. Version initiale : sans option

Écrire une fonction Python *prix_simple(poids : float) → float* qui prend en paramètre le poids du colis (en grammes) et qui renvoie son tarif de base (en euros) en respectant les règles ci-dessus.

```
assert prix_simple(150) == 4
assert prix_simple(500) == 7
assert prix_simple(1200) == 12
```

```
def prix_simple(poids):
    if poids <= 250:
        return 4
    elif poids <= 1000:
        return 7
    else:
        return 12
```

2. Version enrichie : avec options Express et International

On ajoute deux options indépendantes et cumulatives à notre système de livraison :

- Si l'option **Express** est activée (paramètre *express* à *True*), un **supplément fixe de 5 €** est ajouté au tarif de base.
- Si la destination est **Internationale** (paramètre *international* à *True*), le prix de livraison calculé (tarif de base + éventuel supplément Express) est **augmenté de 50 %** (c'est-à-dire multiplié par 1.5).

Écrire une fonction Python *prix_options(poids, express, internationale)* qui prend en paramètres le poids (int), l'option express (bool) ainsi que l'option internationale (bool) et qui renvoie le prix total de la livraison.

```
# Colis de 150g (4 euros de base) + Pas d'Express + National
assert prix_options(150, False, False) == 4.0

# Colis de 500g (7 euros de base) + Express (+5 euros) + National
assert prix_options(500, True, False) == 12.0

# Colis de 1200g (12 euros de base) + Pas d'Express + International (12 * 1.5)
assert prix_options(1200, False, True) == 18.0

# Colis de 500g (7 euros de base) + Express (+5 euros) + International ((7 + 5) * 1.5)
assert prix_options(500, True, True) == 18.0
```

```
def prix_options(poids, express, international):  
    # Appel de la fonction précédente pour obtenir le tarif de base  
    tarif = prix_simple(poids)  
  
    # Étape 1 : Ajout du supplément Express  
    if express == True: # "if express:" est également tout à fait correct  
        tarif = tarif + 5  
  
    # Étape 2 : Application de la majoration internationale sur le cumul précédent  
    if international:  
        tarif = tarif * 1.5  
  
    return tarif
```

3. Version globale : traitement d'une liste de colis sans option

L'entreprise souhaite maintenant traiter un lot contenant plusieurs colis d'un coup, sans option express ni internationale

Écrivez une fonction **total_lot** qui prend en paramètre une liste contenant les poids de plusieurs colis à envoyer (`list`) et qui renvoie la somme totale à payer pour l'ensemble de ces colis.

```
assert total_lot([150, 500, 1200]) == 23 # car 4 + 7 + 12  
assert total_lot([100, 200]) == 8       # car 4 + 4  
assert total_lot([]) == 0                # lot vide
```

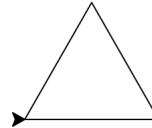
```
def total_lot(liste_poids):  
    somme_totale = 0 # Initialisation de l'accumulateur  
  
    # Parcours par élément de la liste  
    for poids_colis in liste_poids:  
        somme_totale = somme_totale + prix_simple(poids_colis)  
  
    return somme_totale
```

Exercice 3 (5 points)

On utilise ici le module *turtle* de Python pour dessiner des figures géométriques.

1. Écrire une fonction *triangle* qui prend en paramètre une longueur (*int*) et qui dessine un triangle équilatéral dont le côté a cette longueur :

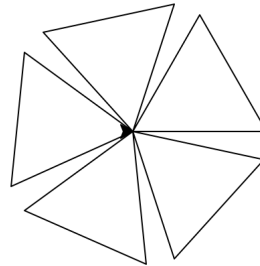
`triangle(100)`



```
def triangle(longueur):  
    for i in range(3):  
        forward(longueur)  
        left(120)
```

2. Écrire une fonction *rotation_triangles* qui prend en paramètres un nombre de triangles (*int*) et une longueur (*int*) et qui dessine la figure ci-dessous :

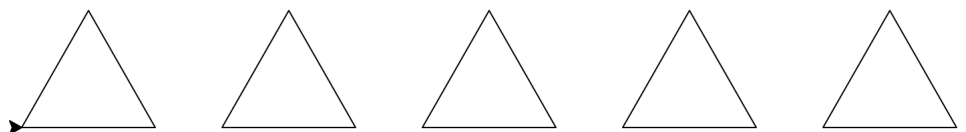
`rotation_triangles(5, 100)`



```
def rotation_triangles(n, longueur):  
    for i in range(n):  
        triangle(longueur)  
        left(360/n)
```

3. Écrire une fonction *ligne_triangles* qui prend en paramètres un nombre de triangles (*int*) et une longueur (*int*) et qui dessine la figure ci-dessous :

`ligne_triangles(5, 100)`



```
def ligne_triangles(n, longueur):  
    distance = longueur // 2  
    for i in range(n):  
        triangle(longueur)  
        penup()  
        forward(longueur + distance)  
        pendown()  
    penup()  
    forward(-n*(longueur+distance))  
    pendown()
```