Informatique de Tronc Commun

Devoir surveillé n°1 (durée : 2 h)

Les réponses sont à donner sur la copie, sauf pour les exercices 3 et 7, où l'on répondra en complétant les tableaux de l'annexe.

Exercice 1. [Exceptions] Erreurs courantes signalées par l'interpréteur

Voici six programmes écrits en Python, dont les lignes ont été ici numérotées :

prog1.py

```
1. L = ["1", 2, 3.0]
2. if L[0] < L[1] and L[1] < L[2]:
3.     print("L strictement croissante")
```

prog2.py

```
1. a, b = 5, 6
2. if a + b = 26:
3.     print(a // b)
```

prog3.py

```
1. n = 4
2. L = [1, 2, 3]
3. L.append[n]
4. L = n * L
```

prog4.py

```
1. a = 2
2. b = 2
3. if a + b > 3:
4.     a = b
5.  else:
6.     b = 0
```

prog5.py

```
1. a, b = 2, 3
2. if a > b:
3.     print(a, "plus grand que , b)
4. else:
5.     print(b, "plus grand que ", a)
```

prog6.py

```
1. a = 0
2. for i in range(5):
3.     b = b + 1
```

Lorsque l'on lance l'exécution de ces six programmes, l'interpréteur retourne une erreur et l'exécution du programme est stoppée.

Voici (de façon désordonnée) les commentaires (complets ou partiels) que l'on peut lire à l'écran, suite à ces tentatives infructueuses d'exécution :

- Message A : `SyntaxError: invalid syntax`
- Message B : `NameError : name 'b' is not defined`
- Message C : `'<' not supported between instances of 'str' and 'int'`
- Message D : `SyntaxError : EOL while scanning string literal`
- Message E : `IndentationError: unindent does not match any outer indentation level`
- Message F : `TypeError: 'builtin_function_or_method' object is not subscriptable`

Pour chaque programme,

1. Indiquer le message d'erreur qui s'affiche, puis :
2. Proposer une correction minimale, de sorte qu'il s'exécute sans déclencher d'erreur, en réécrivant uniquement la(les) ligne(s) modifiée(s) et son(leur) numéro(s).

Exercice 2. Types de base – opérations ; modules usuels

1. Définir une variable entière, `duree`, dont la valeur est 5678, représentant une durée en secondes.
Écrire un nombre minimal d'instructions permettant d'afficher la durée en heures, minutes et secondes.
On définira trois variables, `nh`, `nm`, `ns`, représentant les nombres d'heures, minutes et secondes, nécessaires à l'affichage. La durée étant de 5678 secondes, l'affichage aura la forme :

« La durée est de 1 heure 34 minute 38 secondes. ».

2. L'opérateur booléen XOR (ou exclusif) est décrit par la table de vérité suivante :

a	b	a XOR b
VRAI	VRAI	FAUX
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

On suppose que deux variables booléennes, a et b, ont été définies et qu'une valeur booléenne leur a été affectée. Définir une variable booléenne c, à l'aide d'une expression booléenne, utilisant les variables a et b et des opérateurs booléens, de sorte que la valeur de c soit égale à a XOR b.

3. Écrire un nombre minimal d'instructions permettant de définir une variable `x` dont la valeur est

$$\cos^2\left(\frac{\pi}{7}\right) + \ln\left(\frac{\pi}{11}\right).$$

Exercice 3. [Programmation impérative] Séquences d'instructions – états de la mémoire

On considère les cinq scripts suivants, indépendants les uns des autres.

prog1.py
```
x = 12
y = x
x = 13
```

prog2.py
```
x, y = 12, 7
x = y
y = x
```

prog3.py
```
x = 12
y = 7
x, y = y, x
```

prog4.py
```
x, y = 12, 7
z = x
x = y
y = z
```

prog5.py
```
x, y = 12, 7
x = x + y
y = x - y
x = x - y
```

1. Compléter le tableau situé en annexe en donnant l'état des variables après exécution de chaque instruction.
Si une variable n'est pas définie, on placera une croix dans la case correspondante.

	prog1.py			**prog2.py**			**prog3.py**			**prog4.py**			**prog5.py**		
État des variables	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
Initialement (Avant exécution de la 1ère instruction)															
Après exécution															
de la 1ère instruction															
de la 2ème instruction															
de la 3ème instruction															
…															

2. Que réalisent les programmes 3, 4 et 5, discuter des avantages et inconvénients potentiels de chacun (répondre en deux-trois lignes maximum).

Exercice 4. Fonctions

1. Définir une fonction, `f(a, b)`, renvoyant le quotient et le reste dans la division euclidienne de a par b (on ne testera pas la stricte positivité de b).

2. Définir une variable `n` égale à 12345 et une variable `p` égale à 67.
Écrire une séquence minimale d'instructions permettant d'afficher, grâce à un appel à la fonction `f` le quotient dans la division euclidienne de 12345 par 67 sous la forme :

« Le quotient dans la division euclidienne de 12345 par 67 est 184 , et le reste est 17 .»

L'affichage devra rester correct si les valeurs des variables `n` et `p` sont **modifiées**.

3. Définir une fonction `g(n)`, dans laquelle on utilisera un ou plusieurs appels à la fonction `f` et renvoyant le nombre de boîtes de six œufs nécessaires pour ranger n œufs. On pourra utiliser un (ou plusieurs branchements conditionnels).

4. Proposer trois tests pour la fonction `g`. On donnera les entrées proposées pour la fonction et les sorties attendues.

Exercice 5. [Structures de contrôle] Branchements conditionnels sans et avec alternative

On considère la fonction `mystere` suivante :

```
def mystere(a):
    x = 0
    if a % 2 == 0:
        x = x + 1
    if a % 3 == 0:
        x = x + 1
    return x
```

1. Que renvoient les appels `mystere(12)`, `mystere(13)` et `mystere(9)` ?

2. Quelles sont toutes les valeurs qui peuvent être renvoyées par la fonction `mystere` ?

3. Décrire précisément et le plus concisément possible les valeurs **en entrée** correspondant à chacune des sorties possibles.

4. Reprendre les questions précédentes (**1.** à **3.**) pour la fonction `mystere2` suivante :

```
def mystere2(a):
    x = 0
    if a % 2 == 0:
        x = x + 1
    elif a % 3 == 0:
        x = x + 1
    return x
```

Exercice 6. [Structures de contrôle] Branchements conditionnels sans et avec alternative

On cherche à résoudre une **équation ayant la forme suivante** :

$$a.x^2 + b.x + c = 0$$

et dont les coefficients sont entiers.

On pourra utiliser dans cet exercice des branchements conditionnels sans alternative (`if … : …`) ou avec alternative (`if … : … else : …`), mais il n'est pas autorisé d'utiliser une structure de la forme `if … : … elif … : …`).

1. On suppose dans un premier temps que $a \neq 0$. Définir une fonction `solutions1` qui prend en argument les 3 entiers a, b et c et qui renvoie sous la forme d'un tuple l'ensemble des solutions réelles de l'équation (la fonction renverra donc, suivant les cas, un tuple vide, `()`, un tuple à un élément , `(x0,)`, ou un tuple à deux éléments, `(x1, x2)`).

2. Reprendre la fonction précédente en une fonction `solutions2`, **faisant appel à la fonction `solutions1`** , et qui renvoie de même l'ensemble des solutions de l'équation mais sans aucune condition sur a.
On convient que la fonction renverra la valeur `None` si l'équation admet une infinité de solutions.

Exercice 7. [Structures de contrôle] Boucles `for`

On considère les trois programmes ci-dessous :

P1.py

```
1. n = 3
2. u = 2
3. s = 0
4. for k in range(n):
5.     u = u + 2
6.     s = s + u
```

P2.py

```
1. n = 3
2. u = 2
3. s = 0
4. for k in range(n):
5.     s = s + u
6.     u = u + 2
```

P3.py

```
1. n = 3
2. u = 2
3. s = 0
4. for k in range(n):
5.     s = s + u
6. u = u + 2
```

1. Compléter les tableaux fournis en annexe dans lesquels on indiquera, à chaque fois, depuis le début jusqu'à la fin de l'exécution du programme :
   a. dans la colonne de gauche, le numéro de ligne de l'instruction exécutée ;
   b. dans les trois colonnes de droite, les valeurs en mémoire des trois variables `u`, `s` et `k` à l'issue de l'exécution de l'instruction référencée dans la colonne de gauche [on mettra une croix dans la case correspondant à la variable si celle-ci n'a pas encore été affectée].

2. Quelles sont les valeurs des variables u et s en fin d'exécution, pour chacun de ces trois programmes si la variable `n` est initialisée (ligne 1) à la valeur 100 (les autres instructions restant inchangées) ?

Exercice 8. [Structures de contrôle] Boucles `while`

Pour toutes les fonctions à écrire dans cet exercice, il est exigé d'utiliser une boucle `while` pour atteindre l'objectif fixé.

1. Écrire une fonction `cubes1` qui prend pour argument un entier positif et qui affiche les cubes d'entiers positifs qui sont inférieurs ou égaux à cet entier.

Exemple d'appel à la fonction :

```
>>> cubes1(50)
0
1
8
27
```

2. Modifier la fonction `cubes1` en une fonction `cubes2` qui prend pour argument un entier positif et qui renvoie la liste des cubes d'entiers positifs qui sont inférieurs ou égaux à cet entier.

Exemple d'appel à la fonction :

```
>>> cubes2(50)
[0,1,8,27]
```

Pour les questions 3. à 6., il est imposé de **ne pas utiliser** de fonction de conversion de type (`int`, *etc.*), ni la division entière (//) ou flottante (/), ni les fonctions `floor` ou `ceil` du module `math`.

3. Écrire la fonction `e1` qui prend pour argument un flottant `x` positif et qui renvoie la liste des entiers positifs inférieurs ou égaux à `x`.

Exemples d'appel à la fonction :

```
>>> e1(7.22)
[0,1,2,3,4,5,6,7]
```

```
>>> e1(2)
[0,1,2]
```

4. Modifier la fonction `e1` en une fonction `e2` qui prend pour argument un flottant `x` positif et qui renvoie le plus grand entier inférieur ou égal à `x`.

Exemples d'appel à la fonction :

```
>>> e2(7.22)
7
```

```
>>> e2(2)
2
```

5. Modifier la fonction `e2` en une fonction `e3` qui prend pour argument un flottant `x`, **de signe quelconque**, et qui renvoie le plus grand entier inférieur ou égal à `x` [fonction `floor` de Python].

Exemple d'appel à la fonction :

```
>>> e3(7.22)
7
>>> e3(-7.22)
-8
```

```
>>> e3(5)
5
>>> e3(-5)
-5
```

6. Modifier la fonction `e3` en une fonction `e4` qui prend pour argument un flottant `x`, **de signe quelconque**, et qui renvoie le plus petit entier **supérieur ou égal** à `x` [fonction `ceil` de Python].

Exemple d'appel à la fonction :

```
>>> e4(7.22)
8
>>> e4(-7.22)
-7
```

```
>>> e4(5)
5
>>> e4(-5)
-5
```

Exercice 9. [Algorithmique] Utilisation d'accumulateurs

1. Définir une fonction nommée `somme1` dont l'argument est une liste de nombres, entiers ou flottants, et qui renvoie la somme des carrés de ces nombres.
La fonction utilisera une boucle `for` et un accumulateur.

Exemple d'appel à la fonction :

```
>>> somme1([1, 2, 3, 4])
30
```

2. Définir une fonction `somme2` dont l'argument est une liste de nombres, entiers ou flottants, et qui renvoie le carré de la somme de ces nombres.

Exemple d'appel à la fonction :

```
>>> somme2([1, 2, 3, 4])
100
```

Exercice 10. [Algorithmique] Utilisation d'accumulateurs

On considère les trois fonctions suivantes, prenant en entrée, une liste de flottants.

```
def mystere_1(L):
    s = 0
    for i in range(len(L)):
        s = s + L[i]
    return s / len(L)
```

```
def mystere_2(L):
    s = 0
    ind = 0
    for i in range(len(L)):
        if L[i] > 0:
            s = s + L[i]
            ind = ind + 1
    if ind != 0:
        return ind, s / ind
```

```
def mystere_3(L):
    s = 0
    ind = 0
    for i in range(len(L)):
        if L[i] < 0:
            s = s + L[i]
            ind = ind + 1
    if ind != 0:
        return ind, s / ind
```

1. Que réalise la fonction `mystere_1` ?

2. Que renvoie l'appel suivant ?

```
>>> mystere_1([0, 2, 1, 3, 4, -1, -2])
```

3. Que réalise la fonction `mystere_2` ?

4. Que renvoie l'appel suivant ?

```
>>> mystere_2([0, 2, 1, 3, 4, -1, -2])
```

5. Que réalise la fonction `mystere_3` ?

6. Que renvoie l'appel suivant ?

```
>>> mystere_3([0, 2, 1, 3, 4, -1, -2])
```

Exercice 11. [Structures de données] Construction de listes

1. Écrire une séquence d'instructions générant une liste `L0` de 10 flottants tirés au hasard, pseudo-aléatoirement, parmi les flottants de l'intervalle [0 ; 1[(on fera appel à la fonction `random` du module `random`). On utilisera la méthode `.append()`.

2. Écrire une fonction `liste_alea_int`, de paramètres n, a, et b, qui renvoie une liste de n entiers tirés aléatoirement dans l'intervalle d'entiers $[\![a\,;b]\!]$. On utilisera une compréhension de listes.

3. Écrire une fonction `copie_liste`, prenant en argument une liste `L` et renvoyant une liste `L1` qui est une copie de la liste `L` en entrée. On initialisera la liste `L1` à une liste vide et on utilisera la méthode `.append()`.

Exercice 12. [Structures de données] Parcours de listes - Recherche dans une liste

1. Écrire une fonction `recherche1` qui prend en argument une liste `L` et une valeur `v` et qui renvoie `True` si la valeur est présente dans la liste `L` et `False` si ce n'est pas le cas.

Exemples d'appels à la fonction :

```
>>> recherche1([1, 10, 5, -6, 0], 5)
True
>>> recherche1(['Bob', 'Alice', 'Ted'], 'Harry')
False
```

2. Écrire une fonction `recherche2` qui prend en argument une liste `L` et une valeur `v` et qui renvoie la liste des indices des occurrences de la valeur `v` dans la liste `L` si la valeur est présente dans `L`.
La liste renvoyée sera une liste vide si la valeur `v` n'est pas présente dans la liste `L`.

Exemples d'appels à la fonction :

```
>>> recherche2([1, 10, 5, -6, 0], 4)
[]
>>> recherche2(['A', 'B', 'C', 'E', 'D', 'C', 'D', 'C'], 'C')
[2, 5, 7]
```

Dans le second exemple, le caractère `'C'` est, de fait, présent dans la liste `L`, aux positions 2, 5 et 7.

Exercice 13. [Problème] Progressions arithmétiques

1) Compléter sur votre copie le script ci-dessous :

```
from random import randint
a = randint(...)
...
print(a,...
if ...
```

de sorte qu'à son exécution :

- **a.** il simule le tirage de trois entiers a, b et c, tirés chacun aléatoirement entre 1 et 6 ;
- **b.** affiche ces trois entiers, a, b et c, dans cet ordre ;
- **c.** affiche le nombre de valeurs identiques obtenues ;
- **d.** lorsque a, b et c, dans cet ordre, forment une progression arithmétique non constante, l'affiche en donnant la raison de cette progression.

Exemples d'exécution :

```
>>>
2 5 1
aucun identique
```

```
>>>
6 4 2
aucun identique
progression arithmétique non constante de raison -2
```

```
>>>
5 5 5
trois identiques
```

```
>>>
6 4 4
deux identiques
```

Note : une progression arithmétique constante (de raison 0 donc) correspond exactement au cas où les trois valeurs sont identiques.

2) On suppose que l'on dispose d'une fonction `ordonne(a, b, c)` renvoyant un triplet (un tuple) composé des trois valeurs de a, de b, et de c rangées dans l'ordre croissant, et dont le code, incomplet, est donné ci-dessous.

```
def ordonne(a, b, c):
    ...
    return ...
```

On reprend le script de la question **1)**, à partir de l'affichage des trois valeurs a, b, et c.

- **a.** Écrire l'instruction permettant, par appel à la fonction `ordonne`, de réaffecter leurs valeurs aux variables a, b, et c, de sorte que désormais $a \leq b \leq c$.
- **b.** Reprendre l**es** question**s 1.c. et 1.d.** précédente**s**, sachant que les valeurs a, b, et c sont désormais rangées dans l'ordre croissant.
- **c.** Compléter le code de la fonction `ordonne`, de la façon la plus concise possible (l'utilisation de la fonction `max` n'est pas autorisé**e**).

Exercice 14. [Problème] Crible d'Ératosthène

L'algorithme du crible d'Ératosthène permet d'obtenir la liste des nombres premiers inférieurs ou égaux à un entier naturel n, $n \geq 2$.
On rappelle qu'un nombre premier est un entier naturel supérieur ou égal à 2 admettant exactement deux diviseurs, 1 et lui-même.

L'algorithme peut être mis œuvre de la façon suivante, pour un entier $n \geq 2$ donné.

- On écrit la liste des entiers naturels, compris entre 2 et n.

Puis, on élimine de cette liste - ici, en les remplaçant par la valeur 0, successivement, les multiples des nombres identifiés comme premiers dans la liste (il s'agit à chaque fois le plus petit entier non nul présent dans la liste, après le dernier entier premier dont on a éliminé les multiples).
Ainsi :

- on élimine les multiples de 2. Le plus petit entier non nul suivant dans la liste est alors 3.
- on élimine les multiples de 3. Le plus petit entier non nul suivant dans la liste est alors 5.
- on élimine les multiples de 5. Le plus petit entier non nul suivant dans la liste est alors 7.

…

On arrête le processus itératif lorsque le plus petit entier non nul suivant dans la liste est inférieur ou égal à $\sqrt{n}$.
Après application de cet algorithme, les nombres premiers inférieurs ou égaux à n sont les entiers non nuls encore présents dans la liste.

1. Écrire une fonction `Linit`, prenant en paramètre un entier $n \geq 2$, et renvoyant la liste des entiers compris entre 2 et n inclus, précédés de deux valeurs nulles.

```
>>> Linit(20)
[0, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

2. Écrire une fonction `annule`, prenant en paramètre une liste `L` et un entier $m \geq 2$, et remplaçant par la valeur zéro tous les éléments d'une liste `L` dont les indices de position sont des multiples de m strictement supérieurs à m.

```
>>> L = Linit(20)
>>> annule(L, 3)
>>> L
[0, 0, 2, 3, 4, 5, 0, 7, 8, 0, 10, 11, 0, 13, 14, 0, 16, 17, 0, 19, 20]
```

3. Écrire une fonction `divmax`, prenant en paramètre un entier naturel n et renvoyant le plus grand entier d, inférieur ou égal à la racine carrée d'un entier n.
On pourra recourir à une fonction ou un opérateur prédéfinis (`int`, `floor` (partie entière), `sqrt`, `**0.5`, `//` ou autre, en effectuant si besoin les importations nécessaires).

```
>>> divmax(19)
4
```

```
>>> divmax(25)
5
```

4. Écrire une fonction `elimine`, prenant en paramètre une liste `L` et une valeur `v`, et renvoyant une nouvelle liste composée uniquement des éléments de `L` non égaux à `v`, sans en changer l'ordre et sans en éliminer aucun.

```
>>> elimine([1, 0, 0, 2, 0, 4, 2], 0)
[1, 2, 4, 2]
>>> elimine([1, 1, 3, 2, 5, 4, 2], 0)
[1, 1, 3, 2, 5, 4, 2]
```

5. Écrire un script dans lequel on applique l'algorithme du crible pour obtenir la liste des nombres premiers inférieurs ou égaux à 1000.
On fera impérativement usage des fonctions précédemment définies autant que faire se peut.

Les premières lignes du script seront impérativement :

```
N = 1000
L = Linit(N)
M = ...
...
for ...
...
```

- *Exemple* d'exécution du script en affichant l'état de la liste `L`

Si l'on cherche la liste des entiers premiers inférieurs ou égaux à `N = 30`, le plus grand entier inférieur ou égal à la racine carrée de 30 ($\sqrt{30} \approx 5{,}48$) est 5, et on a pour la liste `L` la suite d'états suivants :

- état initial de la liste :

```
[0, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30]
```

- après élimination des multiples de 2 :

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0, 13, 0, 15, 0, 17, 0, 19, 0, 21, 0, 23, 0, 25, 0, 27, 0, 29, 0]
```

- après élimination des multiples de 3 :

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0, 0, 0, 23, 0, 25, 0, 0, 0, 29, 0]
```

- après élimination des multiples de 5 :

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0]
```

La liste des entiers premiers inférieurs ou égaux à 30 que l'on obtient alors est :

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

ANNEXES

Nom :
Prénom :
Classe :

ANNEXE 1 (exercice 3)

	prog1.py			**prog2.py**			**prog3.py**			**prog4.py**			**prog5.py**		
État des variables	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
Avant exécution de la 1ère instruction															
Après validation															
de la 1ère instruction															
de la 2ème instruction															
de la 3ème instruction															
…															

TOURNER LA PAGE

ANNEXE 2 (exercice 7)

Tableau à compléter :

[On a ci-dessous commencé à remplir le tableau de façon conforme aux consignes]

N° de ligne de l'instruction exécutée	P1.py			N° de ligne de l'instruction exécutée	P2.py			N° de ligne de l'instruction exécutée	P3.py		
	Valeurs des différentes variables après exécution de l'instruction				Valeurs des différentes variables après exécution de l'instruction				Valeurs des différentes variables après exécution de l'instruction		
	u	***s***	***k***		***u***	***s***	***k***		***u***	***s***	***k***
Avant exécution de la première instruction				Avant exécution de la première instruction				Avant exécution de la première instruction			
1				1				1			
2	2			2	2			2	2		
3				3				3			
4				4				4			
5				…				…			
4											
…											