Informatique de Tronc Commun

Devoir surveillé n°1 (durée : 2 h)

Les réponses sont à donner sur la copie, sauf pour l'exercice 6, 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.   def h(x, y):
2.        if x < y:
3.           return x
4.       return y
```

prog2.py

```
1. L = []
2. for i in range(10):
3.     L = L.append(1)
```

prog3.py

```
1. t = (2, 3)
2. if t(1) < 0:
3.     print(t)
```

prog4.py

```
1. i = 0
2. for i in range(10):
3.     a = i + a
```

prog5.py

```
1. n, L = 10, []
2. for i in range(n / 2):
3.     L.append(i)
```

prog6.py

```
1. def f(2, 3):
2.     return a + b / 3
```

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 'a' is not defined`
- Message C : `AttributeError: 'NoneType' object has no attribute 'append'`
- Message D : `TypeError: 'tuple' object is not callable`
- Message E : `TypeError: 'float' object cannot be interpreted as an integer`
- Message F : `IndentationError: unindent does not match any outer indentation level`

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 ; fonctions

1. Quelle instruction écrire pour définir une variable entière, `dmy`, dont la valeur est 12052004, représentant une date de naissance.
2. La variable `dmy` étant définie, on exécute les instructions suivantes :

```
a = dmy % 10
b = dmy // 10
```

Donner les valeurs de `a` et `b`.

3. Définir une fonction `majeur(dn, annee)`, prenant en entrée un entier `dn`, susceptible de représenter une date de naissance, et un entier `annee`, représentant l'année en cours, et renvoyant un booléen, `True` ou `False`, selon que la date de naissance est celle d'une personne majeure ou non.
On devra obtenir par exemple :

```
>>> majeur(12052004, 2023)
True
```

4. Que dire du test suivant ?

```
>>> majeur(05122006, 2023)
```

Exercice 3. Modules usuels

1. Quelle(s) instruction(s) écrire pour pouvoir calculer le carré du cosinus de $\frac{\pi}{5}$?

2. **a.** Définir une fonction `test(x)`, prenant en argument un flottant x, et renvoyant un booléen selon que $e^{\ln x}$ est égal à x ou non. On écrira les importations nécessaires.
b. Que penser du résultat suivant ? (répondre en une phrase).

```
>>> test(123456789.0)
False
```

3. On rappelle la documentation de la fonction `random` définie dans le module `random` :

```
>>> help(random)
Help on built-in function random:

random() method of random.Random instance
    random() -> x in the interval [0, 1).
```

a. Considérant un réel x de l'intervalle $[0; 1[$, et deux réels a et b, $a < b$, quel encadrement peut-on donner pour la valeur $(b-a)x+a$?

b. Définir une fonction `uniform(a, b)`, prenant en entrée deux flottants a et b et renvoyant un flottant aléatoire, compris dans l'intervalle $[a; b[$.

Exercice 4. Branchements conditionnels

1. On considère les fonctions suivantes :

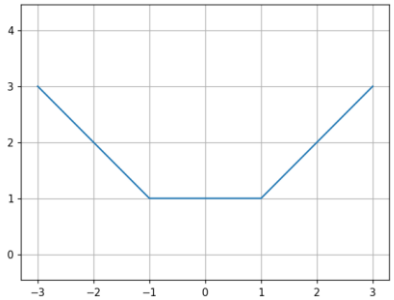
```
def f(x):
    if x < x ** 2:
        return 0
    elif x < 0:
        return 1
```

```
def g(x):
    if x > x ** 2:
        return 0
    elif x < 0:
        return 1
    else:
        return 2
```

a. Que renvoient les appels `f(2)`, `f(0.5)`, `f(-1)` ?

b. Que renvoient les appels `g(2)`, `g(0.5)`, `g(-1)` ?

2. **a.** Définir une fonction `h(x)` effectuant un minimum de comparaisons et implémentant la fonction h définie sur $\mathbb{R}$ et dont la représentation graphique est la suivante (où les droites sont de pente 1 ou -1) :



b. Combien de comparaisons sont effectuées pour calculer `h(-2)`, `h(0)`, `h(2)` ?

Exercice 5. [Programmation impérative] Boucles `while`

On rappelle que la fonction `randint` du module `random`, lorsqu'on l'appelle avec pour arguments deux valeurs entières a et b, avec $a \leq b$, permet de simuler le tirage aléatoire d'un entier parmi les entiers de l'intervalle $[a\ ;\ b]$.

1. On considère le script suivant :

```
from random import randint

a, b = 0, 0
while a != 2 or b == 2:
    a = randint(0, 10)
    b = randint(0, 10)
print(a, b)
```

a. Donner deux affichages distincts qu'il est possible d'obtenir en fin d'exécution.

b. Donner deux affichages distincts qu'il est impossible d'obtenir en fin d'exécution.

2. a. Écrire un script permettant de simuler le tirage simultané de trois dés équilibrés à six faces, jusqu'à obtenir un triple six.

b. Quelle modification apporter au programme pour que les tirages soient répétés jusqu'à ce qu'au moins deux dés affichent un six.

Exercice 6. Boucles while – états de la mémoire

On rappelle que l'évaluation des expressions "`a//b`" et "`a%b`" renvoient en Python, respectivement, la valeur du quotient et du reste dans la division euclidienne d'un entier `a` par un entier `b`, non nul.

On considère le programme ci-dessous, écrit en langage Python et dont on a numéroté les lignes :

```
1.  n = 600
2.  m = 0
3.  d = 2
4.  while d <= n:
5.      if n % d == 0:
6.          m = m + 1
7.          n = n // d
8.      else:
9.          if m > 0 :
10.             print(d, m)
11.             m = 0
12.         d = d + 1
13. print(d, m)
```

On exécute ce programme avec la valeur 600 pour `n`.

Compléter le tableau donné en annexe (à rendre avec la copie), dans lequel **on indiquera, à la fin de l'exécution de chaque itération de la boucle `while`** :

- dans la colonne de gauche, le nombre d'itérations de la boucle `while`
- dans les trois colonnes du milieu, les valeurs des trois variables `n`, `d` et `m` à l'issue de l'exécution de la $k^{\text{ème}}$ itération et dans la colonne de droite, les affichages produits au cours de cette itération.
  [on mettra une croix dans la case correspondant à l'affichage si rien ne s'affiche].

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

1. On considère la fonction suivante :

```
def f(n):
    s = 0
    for i in range(2 * n):
        s = s + i ** 2
    return s
```

Que renvoie l'appel `f(2)` ?

2. Définir une fonction `g`, de paramètre n ($n \in \mathbb{N}$), calculant et renvoyant la somme des entiers impairs compris entre 1 et n inclus.

3. Définir une fonction `h`, de paramètre n, calculant et renvoyant le produit des entiers de 1 à n, inclus.

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

On rappelle les deux opérations suivantes possibles sur les chaînes de caractères :

- Opération de concaténation :
  `ch1 + ch2` crée une nouvelle chaîne qui est la concaténation (mise bout à bout) des chaînes `ch1` et `ch2`
- Opération de duplication :
  `n * ch` crée une nouvelle chaîne qui est la concaténation de n copies de la chaîne `ch`.

On obtiendra par exemple :

```
>>> 'ab' + 'cde'
'abcde'
>>> 3 * 'ab'
'abababb'
```

```
>>> 'ab' + ' ' + 'ba'
'ab ba'
>>> 3 * ' ' + 2 * 'ba'
'   baba'
```

1. Écrire sur la copie les affichages produit par l'exécution du script suivant :

```
n, m = 3, 5
for i in range(n):
    print(m * '+')
```

2. Écrire un script de deux lignes, permettant d'obtenir, en utilisant une boucle `for`, les affichages suivants :

```
x
xx
xxx
xxxx
```

3. Écrire un script de deux lignes, permettant d'obtenir, en utilisant une boucle `for`, les affichages suivants :

```
*****
****
***
**
*
```

4. Écrire sur la copie les affichages produit par l'exécution du script suivant :

```
n = 3
for i in range(n):
    for j in range(i, 2 * i) :
        print(j * '+')
```

5. Écrire sur la copie les affichages produit par l'exécution du script suivant :

```
n, m = 2, 3
for i in range(n):
    for j in range(m):
        print(i * '+' + (m - j) * 'o')
```

Exercice 9. Listes

1. On considère les fonctions suivantes :

```
def f(L):
    L1 = []
    for i in range(len(L)):
        L1.append(L[i] ** 2)
    return L1
```

```
def g(L):
    for i in range(len(L)):
        L[i] = L[i] ** 2
    return L
```

a. Que renvoie l'appel `f([1, 2, 3])` ? et l'appel `g([1, 2, 3])` ?

b. Que fait chacune de ces deux fonctions ? Qu'est-ce qui les différencie dans leur principe ?

2. Écrire une fonction `copie(L)`, créant et renvoyant la copie d'une liste L.

3. Écrire une fonction `somme(L)`, calculant et renvoyant la somme des termes d'une liste de nombres L.

4. Écrire une fonction `h(L)`, prenant en argument une liste d'entiers relatifs et remplaçant les entiers négatifs de cette liste par zéro.

Exercice 10.

On considère les deux fonctions suivantes :

```
def mystere1(L):
    """En entrée : une liste non vide
    d'entiers et en sortie : ???"""
    m = L[0]
    im = 0
    for i in range(len(L)):
        if L[i] > m :
            m = L[i]
            im = i
    return m, im
```

```
def mystere2(L):
    """En entrée : une liste non vide
    d'entiers et en sortie : ???"""
    m = L[0]
    im = 0
    for i in range(len(L)):
        if L[i] >= m :
            m = L[i]
            im = i
    return m, im
```

1. Que renvoie l'appel `mystere1([6,5,3,6,3,4])` ?

2. Que renvoie l'appel `mystere2([6,5,3,6,3,4])` ?

3. Décrire en une (ou deux) phrase(s) ce que renvoient ces deux fonctions.

Exercice 11. Suite de Syracuse

On appelle « suite de Syracuse » de premier terme N ($N > 0$), la suite d'entiers définie par récurrence, de la manière suivante :

$$u_0 = N, \text{ et, pour tout entier } n \geq 0,\ u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

On admet ici la conjecture dite « de Syracuse », selon laquelle quelle que soit la valeur de l'entier N, $N > 0$, il existe un indice n tel que $u_n = 1$.

1. Définir une fonction `f` qui prend en argument un entier a et qui retourne la valeur $\frac{a}{2}$ si a est pair, et la valeur $3a + 1$ si a est impair.

2. Définir une fonction `syracuse`, utilisant la fonction `f`, qui prend en argument les entiers N et n et renvoie le terme de rang n de la suite de Syracuse.

3. Définir une fonction `syracuse_vol`, utilisant la fonction `f`, qui prend en argument l'entier N et renvoie le temps de vol de la suite de Syracuse associée à N, c'est-à-dire le plus petit indice n tel que $u_n = 1$.

ANNEXE à rendre avec la copie **(Exercice 6)**

Nom :
Prénom :
Classe :

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

A la fin de l'itération n°	**Valeurs des différentes variables**			**Affichages**
	n	d	m	
0 **(avant la 1ère itération)**	**600**	**2**	**0**	
1				
2				
…				