Des types, des opérateurs et de la structuration

INFORMATIQUE COMMUNE - Devoir nº 1 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

R Les parties A,B et C sont indépendantes. Il est **fortement** conseillé de les aborder dans l'ordre. Le langage Python est le seul langage informatique autorisé dans les réponses. On s'appliquera à bien respecter les indentations.

A Logique

A1. Une voie du périphérique de Brest est réservée pour certains usages : pour avoir le droit de l'emprunter en voiture, il faut que le véhicule soit un véhicule d'intérêt général ou que le véhicule effectue du covoiturage et, dans tous les cas, que le conducteur ne soit pas un apprenti.

(a) Quel type de variable peut représenter le fait qu'un conducteur est apprenti ou non ?

Solution : un booléen (`bool`), un type de données qui ne peut prendre que deux valeurs : `True` ou `False`

(b) Définir et initialiser arbitrairement trois variables de ce type nommées `general`, `covoiturage` et `apprenti`.

Solution :

```python
apprenti = True
general = False
covoiturage = True
```

(c) Définir une variable `autorise` qui statue sur le fait que la circulation sur la voie réservée est possible ou non.

Solution :

```python
autorise = not apprenti and (general or covoiturage)
```

A2. Une régate est organisée dans la rade de Brest. Pour participer, un candidat doit vérifier les critères suivants :

1. avoir un permis côtier valide,
2. avoir au moins 16 ans,
3. diposer d'un voilier ou le louer à l'organisation de la régate mais dans ce cas il faut avoir au moins 21 ans,
4. les mineurs (strictement moins de 18 ans) ne peuvent participer que s'ils disposent une autorisation parentale.

Calculer la variable booléenne `participe` qui statue sur la participation d'un candidat à la régate. Vous utiliserez exclusivement des variables nommées `age`, `permis`, `autorisation` et `voilier` en précisant leur type et ce qu'elles signifient.

Solution :

```python
permis = True
age = 22
autorisation = False
voilier = False

participe = permis and age > 15 and (voilier or age > 20)  and (autorisation or age > 17)
```

A3. Sur le site d'un cinéma, on peut lire les informations suivantes :

- 8,50 : tarif plein
- 6,50 : tarif sénior (plus de 60 ans)
- 5,50 : étudiant
- 6 : tarif social (famille nombreuse et demandeur d'emploi)
- 4,50 : moins de 18 ans
- 4 : moins de 14 ans

Les inégalités sont à prendre au sens strict : moins de 18 ans, c'est au plus 17 ans.

(a) Écrire une fonction de signature `tarif(age: int, etudiant: bool, social: bool)-> float` qui renvoie le tarif payé selon les paramètres transmis. La fonction renverra toujours le tarif le plus intéressant. Par exemple, un sénior demandeur d'emploi payera 6 €ou un étudiant de 17 ans payera 4,5 €.

Solution :

```python
def tarif(age: int, etudiant: bool, social: bool) -> float:
    if age < 14:
        return 4.0
    elif age < 18:
        return 4.5
    elif etudiant:
        return 5.5
    elif social:
        return 6.0
    elif age > 60:
        return 6.5
```

```
        else:
            return 8.5
```

(b) Donner un exemple d'usage de cette fonction (appeler cette fonction) et récupérer le résultat dans une variable.

Solution :

```
    t = tarif(64, False, False)
```

(c) On souhaite écrire quelques tests afin de vérifier le bon fonctionnement de la fonction `tarif`. À l'aide du mot-clef `assert`, vérifier que si le booléen `etudiant` est vrai, alors la fonction renvoie le bon résultat

Solution :

```
assert tarif(23, True, False) == 5.5
```

A4. On souhaite calculer la somme des n premiers entiers.

(a) Quel est le résultat de $\sum_{k=1}^{10} k$?

Solution : $n(n+1)/2 = 55$

(b) On dispose de la fonction suivante :

```
    def somme(n):
        acc = 0
        for k in range(1, n):
            acc = acc +k
        return acc
```

Quel est le résultat de `somme(10)` ?

Solution : `somme(10)` renvoie 45. Or la somme des 10 premiers entiers vaut $n(n+1)/2$, c'est-à-dire dans notre cas 55. Donc le résultat est faux.

(c) Que faudrait-il modifier dans le code de la fonction `somme` pour que le résultat soit correct ?

Solution : Le `range` s'arrête à $n-1$. Il faut donc écrire `range(1,n+1)`.

```
def somme(n):
    acc = 0
    for k in range(1, n+1):
        acc = acc + k
    return acc
```

B Suites

B1. Écrire les instructions Python nécessaires pour importer la fonction `sin` et la constante `pi` de la bibliothèque `math` afin de pouvoir s'en servir comme suit : `sin(pi/2)`.

Solution :

```
from math import sin, pi
```

R Dans toute la suite de l'examen, on considère que toutes les fonctions mathématiques nécessaires sont correctement importées comme dans votre réponse ci-dessus.

B2. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie de manière explicite par

$$u_n = 2^{n-1}\sin\left(\frac{\pi}{2^{n-1}}\right)$$

Écrire une fonction de signature `u(n: int)—> float` qui renvoie la valeur u_n.

Solution :

```
def u(n):
    dpn = 2**(n—1)
    return dpn * sin(pi/dpn)
```

B3. On considère la suite $(v_n)_{n\in\mathbb{N}}$ définie de manière explicite par

$$v_n = \sum_{k=0}^{n}(-1)^k 2^{-k}$$

Écrire une fonction de signature `v(n: int)—> float` qui renvoie la valeur v_n.

Solution :

```
def v(n):
    s = 0
    for k in range(n+1):
        s += (—1)**k * 2**(—k)
    return s
```

B4. On considère la suite de Tribonacci[1] $(t_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} 0 & \text{si } n = 0 \\ 0 & \text{si } n = 1 \\ 1 & \text{si } n = 2 \\ t_n = t_{n-1} + t_{n-2} + t_{n-3} & \text{sinon} \end{cases} \tag{1}$$

1. Le T majuscule n'est pas obligatoire !

(a) Écrire une fonction de signature `t(n: int)-> int` qui renvoie la valeur t_n.

Solution :

```
def t(n):
    if n == 0 or n==1:
        return 0
    elif n == 2:
        return 1
    else:
        u0 = 0
        u1 = 0
        u2 = 1
        u3 = 0
        for k in range(3,n+1):
            u3 = u2 + u1 + u0
            u0 = u1
            u1 = u2
            u2 = u3
        return u3
```

R Dans la suite de l'examen, il est possible d'appeler les fonctions précédemment définies.

(b) On souhaite calculer la limite du rapport de deux termes consécutifs de la suite $(t_n)_{n\in\mathbb{N}}$, c'est-à-dire $\lim_{n\to+\infty} \frac{t_n}{t_{n-1}}$. Écrire une fonction de signature `limit(n:int)-> float` qui renvoie la valeur $\frac{t_n}{t_{n-1}}$. On garantira par une assertion que l'entier `n` fourni en paramètre d'entrée est strictement supérieur à 2. Quel est l'intérêt de ce mécanisme ?

Solution : Sans l'assertion et si n vaut 1 ou 2, la fonction déclenchera une exception de type `ZeroDivision` et arrêtera le programme, ne sachant plus faire !

```
def limit(n):
    assert n > 2
    return t(n) / t(n-1)
```

(c) On souhaite vérifier expérimentalement la limite de $\frac{t_n}{t_{n-1}}$ quand n tend vers l'infini. Notre conjecture est que cette limite vaut $1,839286755214161$. On souhaite tester cette conjecture jusqu'à une erreur absolue de $0,00001$, c'est-à-dire $|1,839286755214161 - \frac{t_n}{t_{n-1}}| \leqslant 0,00001$. Écrire une fonction de signature `conjecture(epsilon: float)-> int` qui renvoie le premier entier n pour lequel le test est vérifié. On pourra utiliser la fonction `abs(x)` qui renvoie la valeur absolue d'un `x`.

Solution :

```
def conjecture(epsilon: float) -> int:
    n = 3
    while abs(1.839286755214161 - limit(n)) > epsilon:
        n += 1
    return n
```

(d) La fonction conjecture renvoie-t-elle toujours un résultat ?

Solution : Oui, si la conjecture est vraie… Sinon, un appel de cette fonction peut ne pas terminer, c'est-à-dire que la boucle `while` peut s'exécuter éternellement, la condition de sortie n'étant jamais validée.

B5. La suite de Kakutani $(K_n)_{n\in\mathbb{N}}$ est définie par K_0 et :

$$K_{n+1} = \begin{cases} \frac{K_n}{2} & \text{si } K_n \text{ est pair} \\ \frac{3K_n+1}{2} & \text{sinon} \end{cases} \qquad (2)$$

Tout comme la suite de Syracuse, cette suite semble atteindre 1 au bout d'un certain temps, quelque soit $K_0 > 1$.

(a) Écrire une fonction de signature `kakutani(K0: int)-> list[int]` qui renvoie la liste des termes de la suite jusqu'à atteindre la valeur 1.

Solution :

```python
def kakutani(K0: int) -> int:
    K = [K0]
    while K0 != 1:
        if K0 % 2 == 0:
            K0 = K0 // 2
        else:
            K0 = (3 * K0 + 1) // 2
        K.append(K0)
    return K
```

(b) Comment pourrait-on en déduire l'indice n pour lequel la suite de Kakutani atteint 1 ?

Solution : En calculant la longueur de la liste résultat.

```python
len(kakutani(1346))
```

C Des listes et des fonctions

C1. Créer une liste Python à 100 éléments ne contenant que des entiers valant -1 en utilisant une boucle `for`.

Solution :

```python
L = []
for _ in range(100):
    L.append(-1)
```

C2. Créer une liste Python contenant les entiers pairs strictement supérieurs à 20 et strictement inférieurs à 100 en utilisant une boucle `for`.

Solution :

```python
L = []
for i in range(22, 100):
    if i % 2 == 0:
        L.append(i)
```

C3. On dispose d'un signal numérique issu d'un oscilloscope numérique sous la forme d'une liste. Écrire une fonction de signature `ecreter(signal: list[float], v_min, v_max)-> list[float]` qui produit un signal écrété : les valeurs strictement inférieures à `v_min` sont remplacées par `v_min` et les valeurs strictement supérieures à `v_max` par `v_max`. Par exemple, `ecreter([-50, 0, 50, 23, -17], -25, 25)` renvoie le signal `[-25, 0, 25, 23, -17]`.

Solution :

```python
def ecreter(signal: list[float], v_min: float, v_max: float) -> list[float]:
    s = []
    for i in range(len(signal)):
        if signal[i] < v_min:
            s.append(v_min)
        elif signal[i] > v_max:
            s.append(v_max)
        else:
            s.append(signal[i])
    return s
```

C4. On souhaite créer un outil pour transposer une mélodie simple, c'est-à-dire la reproduire dans une tonalité plus grave ou plus aigüe. Les notes sont représentées par une liste de chaînes de caractères. On se donne une gamme, la gamme chromatique ascendante qui est la base d'écriture de nos mélodies :

```python
gas = ['Do', 'Do#', 'Ré', 'Ré#', 'Mi', 'Fa', 'Fa#', 'Sol', 'Sol#', 'La', 'La#', 'Si']
```

Un octave (par exemple du do au do supérieur) est composé de **douze** demi-ton. Toutes les mélodies considérées dans cet exercice sont descriptibles par les notes de cette gamme, c'est-à-dire qu'il n'est pas nécessaire de changer d'octave.

■ **Exemple 1 — Transposition.** La mélodie `['Do', 'Ré', 'Mi', 'Fa', 'Sol', 'Sol', 'Do']` transposée de trois demi-tons vers l'aigü dans la gamme chromatique devient : `['Ré#', 'Fa', 'Sol', 'Sol#', 'La#', 'La#', 'Ré#']`. Si on transpose cette dernière mélodie de trois demi-tons vers le grave, on retrouve la mélodie dans le ton original.

(a) Écrire une fonction de signature `indice(gamme: list[str], note : str)-> int:` qui renvoie l'indice de la note dans la gamme. Si la note n'existe pas dans la gamme, la fonction renvoie `None`. Par exemple, `indice(gas, 'Mi')` renvoie `4`.

Solution :

```python
def indice(gamme: list[str], note : str) -> int:
    for i in range(len(gamme)):
        if note == gamme[i]:
            return i
    return None
```

(b) Écrire une fonction de signature `transpose_note(gamme: list[str], note : str, intervalle: int)-> str` qui renvoie la note transposée d'après un intervalle positif (vers l'aigü) ou négatif (vers le grave) spécifié en demi-tons. Par exemple, `transpose_note(gas, 'Do', 3)` renvoie `'Ré#'` et `transpose_note(gas, 'Ré#', -3)` renvoie `'Do'`.

Solution :

```python
def transpose_note(gamme, note, intervalle):
    i = indice(gamme, note)
    nouvel_indice = (i + intervalle) % 12 # 12 demi-tons dans une octave
    return gamme[nouvel_indice]
```

(c) Écrire une fonction de signature `transpose_melodie(gamme: list[str], melodie: list[str], intervalle : int)-> list[str]` qui renvoie la mélodie transposée dans la gamme selon un intervalle spécifié en demi-tons (positif ou négatif, cf. exemple 1).

Solution :

```python
def transpose_melodie(gamme, melodie, intervalle):
    m = []
    for i in range(len(melodie)):
        m.append(transpose_note(gamme,melodie[i], intervalle))
    return m
```

(d) Écrire une fonction de signature `identiques(gamme, m1, m2)` qui renvoie `True` si les deux mélodies `m1` et `m2` sont identiques moyennant une transposition dans la gamme et `False` sinon.

Solution :

```python
def identiques(gamme, m1, m2):
    if len(m1) != len(m2):
        return False
    else:
        for intervalle in range(12):  # 12 demi-tons possibles
            tm1 = transpose_melodie(gamme, m1, intervalle)
            if m2 == tm1:
                return True
        return False
```