Des types, des opérateurs et de la structuration

INFORMATIQUE COMMUNE - Devoir n° 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.

A Fiesta

A1. Pour des raisons d'incompatibilité, une fête avec Alix, Brieg et Coline ne peut avoir lieu que sous certaines conditions : soit Alix et Brieg sont présents mais pas Coline ; soit Coline est présente mais pas Alix ni Brieg.

(a) Quel type de variable peut représenter le fait qu'Alix veut aller à la fête ou non en Python ?

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

(b) On représente les invités potentiels par leur initiale `a` pour Alix, `b` pour Brieg, `c` pour Coline. Connaissant la volonté de chacun, exprimer en Python la condition qui permet de statuer sur la présence à une fête d'Alix, Brieg ou Coline.

> **Solution :** `a and b and not c or not a and not b and c`

A2. On souhaite compter le nombre de participants à une fête. On dispose d'un détecteur à l'entrée capable de discerner les animaux de compagnie, les fêtards et les parents qui veulent surveiller leur progéniture. Ce détecteur se traduit par une fonction Python dont le prototype est `detecter()` et qui renvoie un entier : `0` s'il s'agit d'un animal, `1` pour un fêtard, `2` pour un parent ou -1 lorsque la détection échoue.

(a) Créer et initialiser quatre variables Python qui représentent le nombre d'animaux, le nombre de fêtards, le nombre de parents et le nombre d'échecs à la détection.

> **Solution :**
>
> ```
> n_animaux = 0
> ```

```
n_fetards = 0
n_parents = 0
n_autres = 0
```

(b) Quel est le type de ces variables ?

Solution : Ce sont des types `int` en Python (par inférence de type).

(c) Appeler la fonction `detecter` et affecter le résultat à la variable `d`.

Solution :

```
d = detecter()
```

(d) Écrire une structure alternative en Python qui permet d'incrémenter les variables définies en (a) en fonction du résultat `d` de la détection. On tiendra compte de tous les cas possibles.

Solution :

```
if d == 0:
    n_animaux += 1
elif d == 1:
    n_fetards += 1
elif d == 2:
    n_parents += 1
else:
    n_autres += 1
```

A3. À la fin de la fête, il faut ranger la maison. Cependant, le rangement ne peut avoir lieu que si les conditions suivantes sont réunies :

- les parents ne sont pas encore arrivés,
- les fêtards ne sont pas partis,
- un seul animal est présent (le chat de la maison).

(a) Écrire une fonction dont le nom est `ranger` et qui permet de statuer sur la possibilité de ranger la maison. Vous préciserez les paramètres formels d'entrée et de sortie de cette fonction ainsi que leur type respectif.

Solution :

```
def ranger(n_animaux: int, n_fetards: int, n_parents: int):
    return n_animaux == 1 and n_fetards > 0 and n_parents == 0
```

Le type de sortie est un booléen.

(b) Donner un exemple d'appel correct de cette fonction en tenant compte de ce qu'elle renvoie.

Solution :

```
ok_pour_ranger = ranger(1, 3, 0)
```

A4. Pour la prochaine fête, vous décidez d'organiser la fermeture des lieux afin d'éviter tout dérapage.

(a) Écrire une fonction de prototype `fermer(participants, litres)` qui renvoie :

- `True` si les participants sont partis ou si le nombre de litres de boisson par participant est strictement supérieur à deux,
- `False` dans les autres cas.

Solution :

```
def fermer(participants: int, litres: int):
    if participants == 0:
        return True
    else:
        return litres / participants > 2
```

(b) À l'aide du mot-clef `assert`, proposer trois assertions pour tester le bon fonctionnement de la fonction `fermer`. On rappelle que cette fonction s'utilise comme suit :

```
assert expression
```

où `expression` une condition. `assert` lève une exception (c'est-à-dire arrête le programme) si cette condition est fausse. L'exécution continue normalement sinon.

Solution :

```
assert fermer(0, 6)
assert not fermer(5, 7)
assert fermer(5, 14)
```

B Suites

B1. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie de manière explicite par

$$u_n = (n-1)2^{n+1} + 2$$

Écrire une fonction de prototype `u(n)` qui renvoie la valeur du terme u_n de la suite.

Solution :

```
def u(n):
    return (n-1)*2**(n+1) + 2
```

B2. On considère la suite $(v_n)_{n\in\mathbb{N}}$ définie par

$$v_n = \sum_{k=0}^{n} k2^k$$

Écrire une fonction de prototype `v(n)` qui renvoie la valeur du terme v_n de la suite.

Solution :

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

B3. On pressent que la valeur de u_n est la même que celle de v_n pour un n donné. Expérimentalement, on aimerait vérifier cette conjecture jusqu'à $n = 20$ inclus. En utilisant une boucle, l'instruction `assert` ainsi que les fonctions `u` et `v`, proposer un test de cette conjecture.

Solution :

```
# TESTS
for i in range(21):
    assert u(i) == v(i)
```

B4. On considère la suite $(s_n)_{n\in\mathbb{N}}$ définie par $s_n = \sum_{k=0}^{n} (-1)^k k$. Écrire une fonction de prototype `s(n)` qui renvoie la valeur de s. Cette fonction utilise l'opérateur `**`.

Solution :

```
def s(n):
    acc = 0
    for k in range(n + 1):
        acc += (-1) ** k * k
    return acc
```

B5. Écrire une fonction de prototype `alt_s(n)` qui renvoie la valeur de s_n. Cette fonction n'utilise pas l'opérateur `**`.

Solution :

```
def alt_s(n):
    acc = 0
    for k in range(n + 1):
        if k % 2 == 0:
            acc +=  k
        else:
```

```
            acc += -k
    return acc
```

B6. Écrire une fonction de prototype `simple_s(n)` qui renvoie $n/2$ si n est pair et $-(n+1)/2$ si n est impair. Cette fonction renvoie un entier.

Solution :

```
def simple_s(n):
    if n % 2 == 0:
        return n // 2
    else:
        return -(n + 1) // 2
```

B7. On considère la suite $(P_n)_{n\in\mathbb{N}^*\setminus\{1\}}$ définie par

$$P_n = \prod_{k=2}^{n} \sqrt{\frac{k^3-1}{k^3+1}}$$

Écrire une fonction de prototype `p(n)` qui renvoie la valeur de P_n. Si $n < 2$, alors la fonction renvoie 1. Ne pas oublier d'importer les fonctions nécessaires à ce calcul.

Solution :

```
from math import sqrt

def p(n):
    acc = 1
    for k in range(2, n + 1):
        acc *= sqrt((k ** 3 - 1) / (k ** 3 + 1))
    return acc

def p(n):
    if n >= 2:
        acc = 1
        for k in range(2, n + 1):
            acc *= sqrt((k ** 3 - 1) / (k ** 3 + 1))
        return acc
    else:
        return 1
```

B8. On considère la suite $(r_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} 0,5 & \text{si } n = 0 \\ 0,1 & \text{si } n = 1 \\ r_n = \frac{\sqrt{r_{n-1}}+\sqrt{r_{n-2}}}{2} & \end{cases} \tag{1}$$

Écrire une fonction de prototype `r(n)` qui renvoie la valeur de r_n.

Solution :

```
from math import sqrt
def r(n):
    r0 = 0.5
    r1 = 0.1
    if n == 0:
        return r0
    elif n == 1:
        return r1
    else:
        for k in range(2, n+1):
            tmp = r0
            r0 = r1
            r1 = (sqrt(r1) + sqrt(tmp))/2
        return r1
```

B9. On souhaite vérifier expérimentalement que la limite de r_n quand n tend vers l'infini est 1. En utilisant une boucle ainsi que le fonction `r`, proposer un test de cette conjecture jusqu'à une erreur relative de $0,00001$, c'est-à-dire $|1-r_n| \leqslant 0,00001$. Votre code permet de trouver l'entier n pour lequel cette condition est vérifiée.

Solution :

```
eps = 0.00001
n = 2
while abs(1 - r(n)) > eps:
    n += 1
print(n, r(n))
```

C Des listes et des fonctions

C1. Créer une liste Python contenant les 20 premiers entiers (en commençant à 0) en utilisant une boucle `for`.

Solution :

```
L = []
for i in range(20):
    L.append(i)
```

C2. Créer une liste Python contenant les entiers impairs strictement inférieurs à 20 en utilisant une boucle `for`.

Solution :

```
L = []
for i in range(20):
    if i % 2 == 1:
        L.append(i)
```

C3. Les années bissextiles comportent 366 jours. Celles-ci n'ont pas nécessairement lieu tous les 4 ans. Pour qu'une année soit bissextile, il faut qu'elle soit divisible par 4, c'est une condition nécessaire mais pas suffisante. Les années qui, de plus, **ne sont pas** divisibles par 100 sont bissextiles. Enfin, les années divisibles par 400 sont bissextiles. Écrire une fonction de prototype `bissextile(annee : int)` qui statue sur le caractère bissextile ou non de l'année passée en paramètre.

Solution :

```
def bissextile(annee):
    if annee % 4 == 0 and not (annee % 100 == 0):
        return True
    elif annee % 400 == 0:
        return True
    else:
        return False

def bissextile(annee):
    return annee % 4 == 0 and (not(annee % 100 == 0) or (annee % 400 == 0))

assert not bissextile(1000)
assert bissextile(2000)
assert bissextile(2016)
assert not bissextile(2022)
for year in [1000, 2000, 2016, 2020, 2022, 2024, 2028, 2030, 2032, 2100, 2400]:
    print(year, bissextile(year))
```

C4. Écrire une fonction de signature `de(xa,ya,xb,yb)` qui renvoie la distance euclidienne (cf. figure 1)de deux points de coordonnées `(xa,ya)` et `(xb,yb)`.

Solution :

```
from math import sqrt

def de(xa,ya,xb,yb):
    return sqrt((xa-xb)**2 + (ya-yb)**2)
```

C5. Écrire une fonction de prototype `dm(xa,ya,xb,yb)` qui renvoie la distance de Manhattan (cf. figure 1) entre deux points de coordonnées `(xa,ya)` et `(xb,yb)`. Mathématiquement, cette distance est définie par : $d_m(a,b) = |x_a - x_b| + |y_a - y_b|$.Le calcul d'une valeur absolue se fait grâce à la fonction `abs` en Python.

Solution :

```
def dm(xa,ya,xb,yb):
    return abs(xa—xb) + abs(ya—yb)
```

C6. Écrire une fonction de prototype `dt(xa,ya,xb,yb)` qui renvoie la distance de Tchebychev (cf. figure 1) entre deux points de coordonnées `(xa,ya)` et `(xb,yb)`. Mathématiquement, cette distance est définie par $d_y(a,b) = \max(|x_a - x_b|, |y_a - y_b|)$. L'usage de la fonction max n'est pas autorisé.

Solution :

```
def dt(xa,ya,xb,yb):
    dx = abs(xa—xb)
    dy =abs(ya—yb)
    if dx >=  dy:
        return dx
    else:
        return dy
```

C7. On dispose d'une liste de points `pts`. Chaque élément de cette liste est un tuple de type `(x,y)`. Écrire une fonction de prototype `distances(pts, d)` qui renvoie une liste qui contient les distances entre tous les points de la liste `pts` selon la distance `d`.

Solution :

```
def distances(points, d):
    dist = []
    for i in range(len(points) — 1):
        xa, ya = points[i]
        for j in range(i + 1, len(points)):
            xb, yb = points[j]
            dist.append(d(xa, ya, xb, yb))
    return dist

pts = [(1, 1), (2, 2), (3, 4), (5, 7), (1, 9)]
result = distances(pts, dm)
print(len(result), result)
```

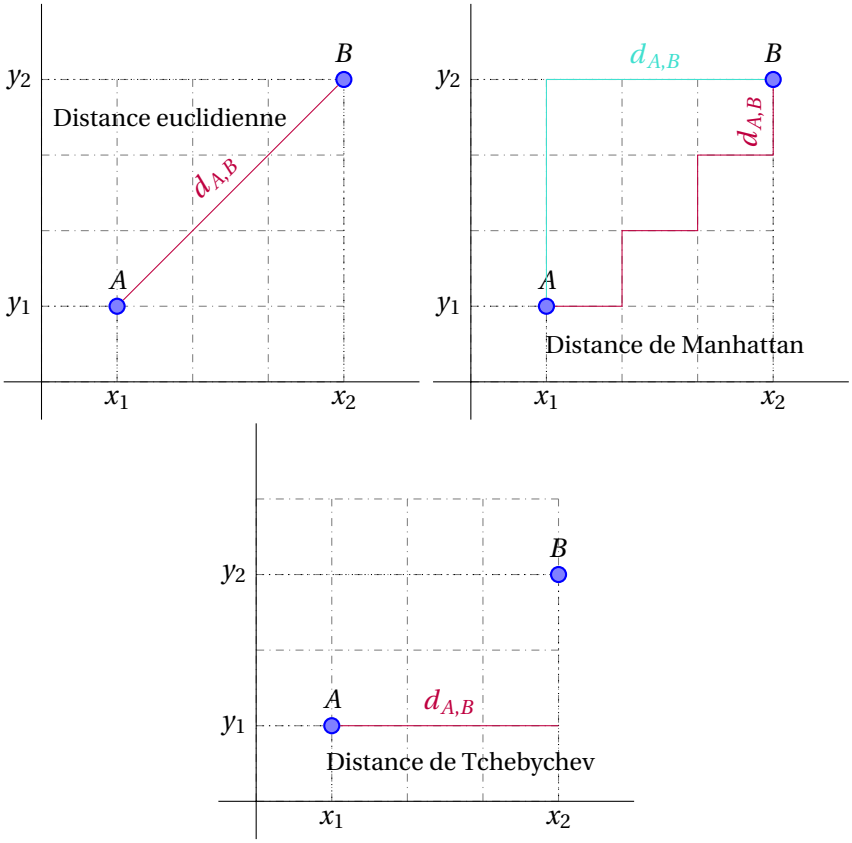

FIGURE 1 – Illustration des différentes distances dans le plan