# DS 1

1 heure et 30 minutes

## Lisez attentivement les consignes ci-dessous

- *Les documents, calculatrices et autres appareils électroniques sont interdits.*
- *La qualité de la rédaction sera prise en compte dans l'évaluation.*
- *Laissez la première page de votre première copie double vierge.*
  - ▷ *N'y écrivez que vos nom et prénom, votre classe, le numéro du DS, la matière.*
  - ▷ *Écrivez-y aussi, en haut à gauche, en gros, en couleur et en l'entourant votre numéro d'élève.*
- *La présentation de la copie sera prise en compte dans l'évaluation.*
  - ▷ *Soignez votre écriture.*
  - ▷ *Maintenez une marge dans vos copies et aérez vos copies.*
  - ▷ *Numérotez vos copies et indiquez sur la première copie le nombre total de copies.*
  - ▷ *Écrivez « OK » sur la première page de votre copie pour montrer que vous avez bien lu ces consignes.*
- ***Vos programmes doivent être clairs.***
  - ▷ *Choisissez judicieusement les noms de vos variables ainsi que les noms de vos fonctions auxiliaires.*
  - ▷ ***Indiquez les indentations par des traits verticaux.***
  - ▷ *Si nécessaire, commentez vos programmes en utilisant une couleur secondaire.*
  - ▷ ***Un programme juste mais qui n'est pas suffisamment clair sera considéré comme faux.***

# Applications

*Dans ce sujet, on étudie une implémentation en* `Python` *de quelques notions de théorie des ensembles relatives aux applications.*

## Partie I – Autour de `append()`

**La commande** `append()`

- *Si* `l` *est une liste et si* `a` *est un objet, on rappelle que la commande* `l.append(a)` *modifie la liste* `l` *en lui ajoutant l'objet* `a`.
- *Par exemple, si*

  `l = [3, 2, -5]`

  *alors, après exécution de la commande* `l.append(50)`, *on aura* `l = [3, 2, -5, 50]`.

**1.** On considère les instructions suivantes :

```
l = []
l.append(1)
l.append(2)
l.append(3)
x = l[2]
```

1 (a) Après exécution de ces instructions, que vaut la variable `l` ?

1 (b) Après exécution de ces instructions, que vaut la variable `x` ?

**2.** On considère les instructions suivantes :

```
m = []
m.append(1)
m.append(2)
m.append(3)
x = m[0]
if m[1] == 1:
    x = x+1
else:
    x = x+10
```

1 Après exécution de ces instructions, que vaut la variable `x` ?

1 **3.** On considère les instructions suivantes :

```
u = []
u.append(1)
u.append(2)
u.append(3)
x = u[0]
y = u[x]
if y == y+1:
    y = y-1
else:
    y = y+1
```

Après exécution de ces instructions, que vaut la variable `y` ?

**4.** On considère la fonction :

```
def f(n):
    l = []
    i = 0
    while i <= n:
        l.append(i)
        i = i+1
    return l
```

1 (a) Que renvoie l'instruction `f(0)` ?

1 (b) Que renvoie l'instruction `f(5)` ?

**5.** On considère la fonction :

```
def g(n):
    l = []
    for i in range(n):
        l.append(i)
    return l
```

1 (a) Que renvoie l'instruction `g(0)` ?

1 (b) Que renvoie l'instruction `g(5)` ?

**6.** On considère les instructions suivantes :

```
w = []
w.append([])
w.append([0])
w.append([0, 1])
w.append([0, 1, 2])
i = 3
x = w[i]
x = x[i-1]
```

1 Après exécution de ces instructions, que vaut la variable `x` ?

7. *Dans cette question, on considère la fonction* `g(n)` *définie à la question* **4.**.
On définit alors la fonction :

```
def h(n):
    l = []
    for i in range(n):
        m = g(i)
        l.append(m)
    return l
```

2 (a) Que renvoie l'instruction `h(0)` ?
1 (b) Que renvoie l'instruction `h(1)` ?
1 (c) Que renvoie l'instruction `h(3)` ?

1,5

8. On considère les instructions suivantes :

```
l = []
l.append(0)
l.append(1)
w = l[1]
x = w[0]
if x == x+1:
    x = x-1
else:
    x = x+1
```

Après exécution de ces instructions, que vaut la variable `x` ?

9. On considère dans cette question les fonctions `g(n)` et `h(n)` définies aux questions **5.** et **7.**.

1 (a) Quand on exécute `g(n)`, combien de fois la commande `append` est-elle appelée ?
3 (b) Quand on exécute `h(n)`, combien de fois la commande `append` est-elle appelée ?
*Le résultat sera simplifié autant que possible.*

## Partie II – Applications

**Objet-fonction associé à une fonction**

*Soit $n \in \mathbb{N}^*$.*

- *Si $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ est une fonction, on lui associe la liste*

$$\texttt{[a}_0\texttt{, a}_1\texttt{, ..., a}_{\texttt{n-1}}\texttt{]}$$

  *où* $\mathtt{a}_0 = f(0)$, $\mathtt{a}_1 = f(1)$, …, $\mathtt{a}_{\mathtt{n-1}} = f(n-1)$.
- *On note alors cette liste* `f` *on l'appelle* objet-fonction associé à $f$.

**Un exemple**

*Par exemple, si $f$ est la fonction de $[\![0, 3]\!]$ dans $[\![0, 3]\!]$ définie par*

$$f(0) = 2, \quad f(1) = 3, \quad f(2) = 1, \quad f(3) = 1,$$

*alors on aura* `f` = `[2, 3, 1, 1]`. *Dans cet exemple, on a $n = 4$.*

**Accéder au « `n` » d'un objet-fonction**

*Dans la suite, si* `f` *est un objet-fonction, on accèdera à l'entier $n$ correspondant à $f$ en utilisant la commande* `taille(f)`, *définie par*

```
def taille(f):
    return len(f)
```

**Remarques**

- *Dans la suite, on pourra identifier*
  - ▷ *l'objet-fonction* `f` *et la fonction $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ ;*
  - ▷ *l'objet* `n` *(qui est de type* `int`*) et l'entier $n \in \mathbb{N}$.*
- *Dans les fonctions prenant un objet-fonction* `f` *en argument, on supposera toujours que* `taille(f)` $\geqslant 1$.

2

**10.** Écrire une fonction `moyenne(f)` qui prend en argument un objet-fonction `f` et qui renvoie sa moyenne.

3

**11.** Écrire une fonction `maximum(f)` qui prend en argument un objet-fonction `f` et qui renvoie son maximum.

3

**12.** Écrire une fonction `estCroissante(f)` qui prend en argument un objet-fonction `f` et qui renvoie `True` quand $f$ est croissante et `False` sinon.

**13.** Écrire une fonction `composee(g, f)` qui prend en argument

- un objet-fonction `f`
- et un objet-fonction `g` tels que `taille(f)` = `taille(g)`

3

et qui renvoie l'objet-fonction associé à $g \circ f$.

*On supposera que* `f` *et* `g`*, donnés en argument, vérifient bien* `taille(f)` = `taille(g)`.

**Définition**

*Si* $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ *est une fonction et si* $x \in [\![0, n-1]\!]$ *est un entier vérifiant* $f(x) = x$*, on dit que* $x$ est un point fixe de $f$.

1 **14.** (a) Écrire une fonction `estPointFixe(f, x)` qui prend en argument un objet-fonction `f` et un objet `x` de type `int` tel que $0 \leqslant$ `x` $\leqslant$ `taille(f)` $- 1$ et qui renvoie

- `True` si $x$ est un point fixe de $f$
- et `False` sinon.

1 (b) Écrire une fonction `pointsFixes(f)` qui prend en argument un objet-fonction `f` et qui renvoie la liste des points fixes de $f$, rangés par ordre croissant.

**15. Question de cours.**

3 Écrire une fonction `indiceElement(l, x)` qui prend en argument une liste `l` triée (*ie* dont les éléments sont classés par ordre croissant) et un nombre `x`, et qui renvoie :

- un indice `i` tel que `l[i]` $=$ `x` s'il en existe un ;
- `-1` si le nombre `x` n'est pas présent dans `l`.

*La fonction* `indiceElement(l, x)` *devra implémenter l'algorithme de recherche par dichotomie présentée en cours.*

**16.** On peut démontrer que toute fonction croissante $f : [\![a, b]\!] \longrightarrow [\![a, b]\!]$ admet un point fixe.

4 Écrire une fonction `pointFixe(f)` qui prend en argument un objet-fonction `f` telle que $f$ est croissante et qui renvoie un point fixe `x` de `f`.

*La fonction* `pointFixe(f)` *devra procéder par dichotomie. On pourra, en plus, procéder par récursivité.*

**Définition**

*Soit* $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$. *On dit que* $f$ est injective *quand*

$$\forall x, y \in [\![0, n-1]\!], \quad x \neq y \implies f(x) \neq f(y).$$

3 **17.** Écrire une fonction `estInjective(f)` qui prend en argument un objet-fonction `f` et qui renvoie `True` quand $f$ est injective et `False` sinon.

## Partie III – Parties

**Notations générales**

*Soit* $n \in \mathbb{N}^*$.

- *Si* $A$ *est une partie de* $[\![0, n-1]\!]$, *on lui associe la liste*

$$\texttt{[b}_0\texttt{, b}_1\texttt{, ..., b}_{\texttt{n-1}}\texttt{]}$$

  - ▷ *où* $\texttt{b}_0 = \texttt{True}$ *si* $0 \in A$ *et* $\texttt{b}_0 = \texttt{False}$ *sinon,*
  - ▷ *où* $\texttt{b}_1 = \texttt{True}$ *si* $1 \in A$ *et* $\texttt{b}_1 = \texttt{False}$ *sinon,*
  - ▷ $\cdots$
  - ▷ *où* $\texttt{b}_{\texttt{n-1}} = \texttt{True}$ *si* $n-1 \in A$ *et* $\texttt{b}_{\texttt{n-1}} = \texttt{False}$ *sinon.*
- *On note alors cette liste* `A`, *et on l'appelle* objet-partie associé à $A$.
- *Dans la suite, on pourra identifier une partie* $A \subset [\![0, n-1]\!]$ *et* `A`, *l'objet-partie associé à* $A$.

**Accéder au « `n` » d'un objet-partie**

- *Dans la suite, si* `A` *est un objet-partie, on accèdera à l'entier* $n$ *correspondant à* $A$ *en utilisant la commande* `tailleAmbiante(A)`, *définie par*

```
def tailleAmbiante(A):
    return len(A)
```

- *Dans les fonctions prenant un objet-partie* `A` *en argument, on supposera toujours que* `tailleAmbiante(A)` $\geqslant 1$.
- *Dans des fonctions prenant deux objets-parties* `A` *et* `B`, *on supposera toujours en outre que* `tailleAmbiante(A)` $=$ `tailleAmbiante(B)`.

**18.** On considère l'objet-partie `A` définie par

$$\texttt{A} = \texttt{[True, True, False, False, True]}$$

1 (a) Quel est le cardinal de la partie $A$ dont elle est l'objet-partie associé ?

1 (b) Quelle est cette partie $A$ ?

1 **19.** On suppose $n = 3$. Donner l'objet-partie associé à $\{1\}$.

2 **20.** Écrire une fonction `cardinal(A)` qui prend en argument un objet-partie `A` et qui renvoie le cardinal de $A$.

3 **21.** Écrire une fonction `estVide(A)` qui prend en argument un objet-partie `A` et qui renvoie `True` si $A$ est vide et `False` sinon.

3 **22.** Écrire une fonction `estInclus(A, B)` qui prend en argument un objet-partie `A` et un objet-partie `B` et qui renvoie `True` si $A \subset B$ et `False` sinon.

2 **23.** (a) Écrire une fonction `intersection(A, B)` qui prend en argument un objet-partie `A` et un objet-partie `B` et qui renvoie l'objet-partie associé à $A \cap B$.

2 (b) Écrire une fonction `union(A, B)` qui prend en argument un objet-partie `A` et un objet-partie `B` et qui renvoie l'objet-partie associé à $A \cup B$.

**Définitions**

*Si* $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ *et si* $A \subset [\![0, n-1]\!]$, *on note*

$$f[A] = \{f(a) \; ; \; a \in [\![0, n-1]\!]\}$$
$$f^{-1}[A] = \{x \in [\![0, n-1]\!] \mid f(x) \in A\}.$$

3

**24.** Écrire une fonction `imageDirecte(f, A)` qui prend en argument un objet-fonction `f` et un objet-partie `A` et qui renvoie l'objet-partie associé à $f[A]$.

*On supposera que si* $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ *alors on a bien* $A \subset [\![0, n-1]\!]$.

3

**25.** Écrire une fonction `imageReciproque(f, A)` qui prend en argument un objet-fonction `f` et un objet-partie `A` et qui renvoie l'objet-partie associé à $f^{-1}[A]$.

*On supposera que si* $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ *alors on a bien* $A \subset [\![0, n-1]\!]$.

4

**26.** Écrire une fonction `listeDesParties(n)` qui prend en argument un entier `n` et qui renvoie la liste des objets-parties associées à toutes les parties $A \subset [\![0, n-1]\!]$.

*On procèdera par récursivité.*

5

**27.** Écrire une fonction `listeDesFonctionsCroissantes(n)` qui prend en argument un entier `n` et qui renvoie la liste des objets-fonctions associées à toutes les $f : [\![0, n-1]\!] \longrightarrow [\![0, n-1]\!]$ croissantes.

FIN DU SUJET.