Groupe	Nom	Prénom

Devoir sur Table 2021 – 2022
Éléments de Programmation I+ – LU1IN011
Durée : 1h30

Documents autorisés: Aucun document ni machine électronique n'est autorisé à l'exception de la carte de référence de Python.

Le sujet comporte 18 pages. Ne pas désagrafer les feuilles.

Répondre directement sur le sujet, dans les boîtes appropriées. La taille des boîtes suggère le nombre de lignes de la réponse attendue. Le quadrillage permet d'aligner les indentations.

Le barème indiqué pour chaque question n'est donné qu'à titre indicatif. Le barème total est lui aussi donné à titre indicatif : 45 points auxquels peuvent s'ajouter des points bonus explicités dans l'énoncé des questions.

La clarté des réponses et la présentation des programmes seront appréciées. Les exercices peuvent être résolus dans un ordre quelconque. Pour répondre à une question, **il est possible, mais pas nécessairement utile, d'utiliser les fonctions qui sont l'objet des questions précédentes, même si vous n'avez répondu à ces questions précédentes.**

Remarque: si nécessaire, on considère que la bibliothèque de fonctions mathématiques a été importée avant les fonctions à écrire. Les types `List` et `Tuple` ont été aussi importés. Sauf mention contraire explicite, seules les primitives Python présentes sur la carte de référence peuvent être utilisées.

Important: **bien qu'implicite, il est toujours nécessaire de donner une définition avec précondition(s) éventuelle(s)**. En revanche, sauf en cas de mention contraire (notamment dans les exercices d'analyse de code), pour les fonctions demandées, la description textuelle et les jeux de tests ne sont *pas* demandés, contrairement aux exercices sur machine.

L'examen est composé de 3 exercices indépendants :

Groupe	Nom	Prénom

Exercice 1 : Points et Vecteurs

Dans cet exercice, on manipule des points du plan, définis comme des couples (abscisse, ordonnée), et des vecteurs, définis commes des couples de points.

On définit donc les alias de type suivants :

```
Point = Tuple[float, float]
Vecteur = Tuple[Point, Point]
```

Pour les tests et les exemples on définit quelques points et vecteur :

```
ori : Point = (0.0, 0.0)

p1 : Point = (0.0, 1.0)
p2 : Point = (1.0, 3.0)
p3 : Point = (-0.5, 0.0)

p4 : Point = (4.5, 0.0)
p5 : Point = (-3.0, 0.0)

nul : Vecteur = (ori, ori)

v1 : Vecteur = (p1, p2)
v2 : Vecteur = (p2, p3)
v3 : Vecteur = (ori, p4)
```

Ainsi `ori` est l'origine du plan, et `v2` est le vecteur allant de $(1, 3)$ en $(-0.5, 0)$.

Question 1.1 : [2/45]

La distance entre deux points du plan de coordonnées (x_1, y_1) et (x_2, y_2) est donnée par la formule :

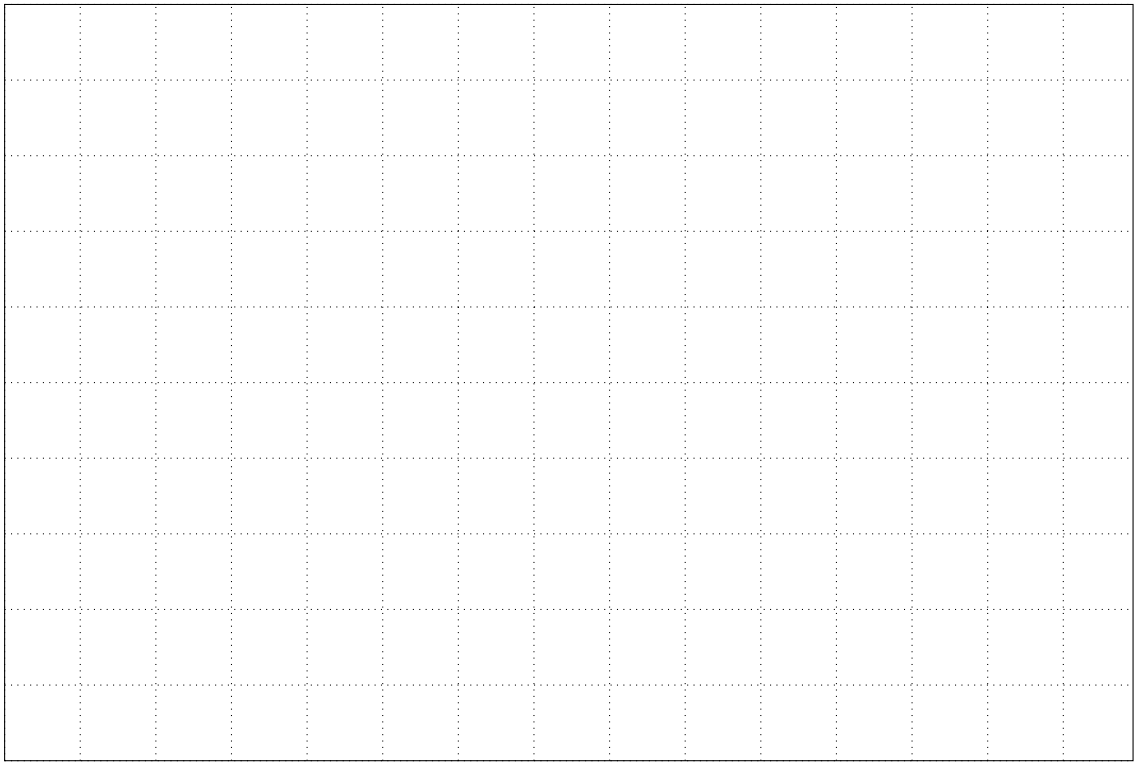
$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ecrire une fonction `distance` qui prend en entrée deux points, et qui renvoie la distance entre ces deux points.

```
>>> distance(ori, p1)
1.0
>>> distance(p1, p1)
0.0
>>> distance(p4, p5)
7.5
>>> abs(distance(ori, (1.0, 1.0)) - math.sqrt(2)) < 10 ** -12
True
```

Remarque. Pour tester que la distance de l'origine au point $(1.0, 1.0)$ vaut bien $\sqrt{2}$, on a utilisé un *encadrement* pour prend en compte les erreurs de calcul avec les flottants.

Groupe	Nom	Prénom

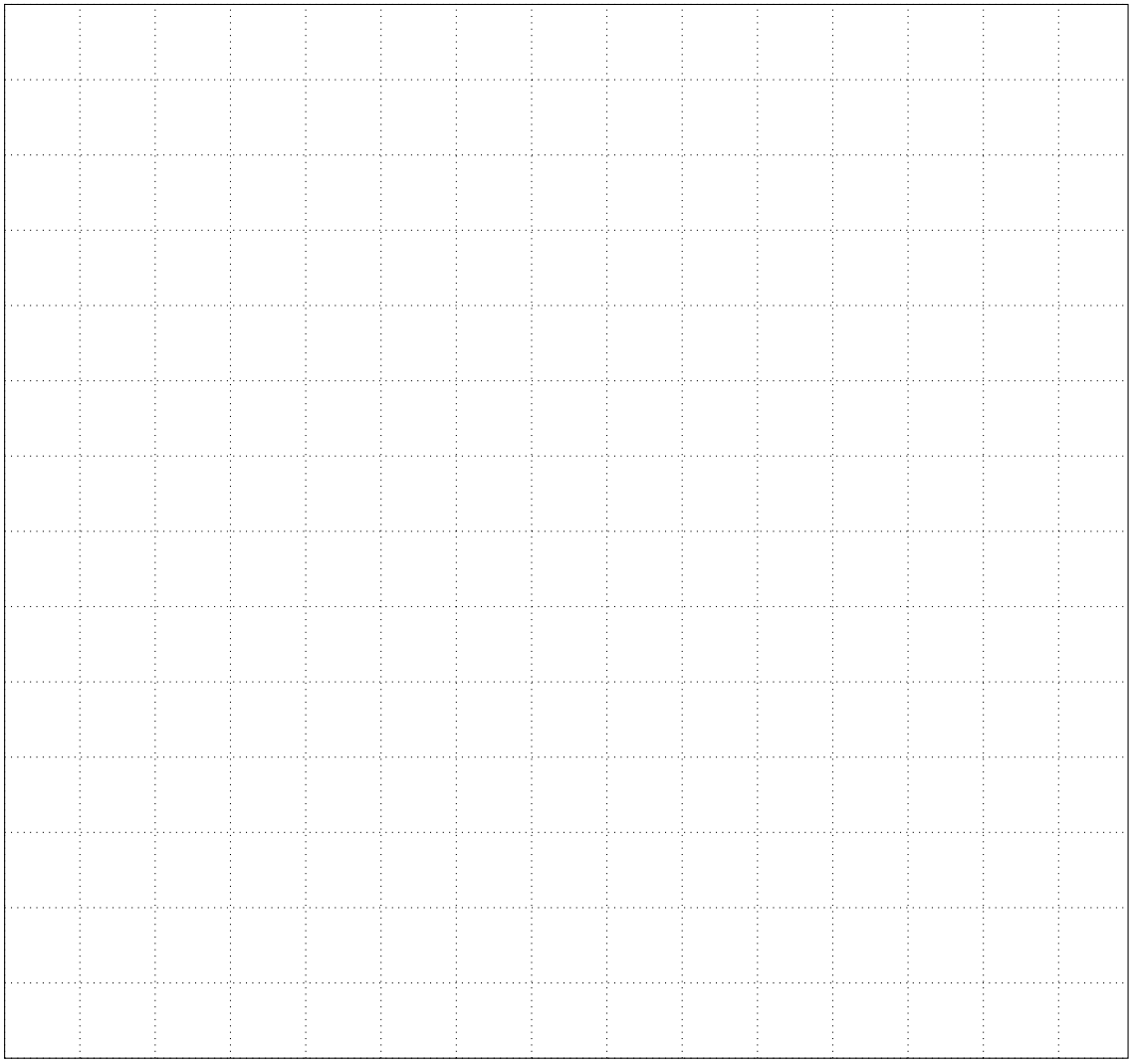


Question 1.2 : [3/45]

Ecire une fonction `longueur` qui prend en entrée une liste de points, et qui renvoie la longueur de la ligne composée de segments, reliant dans l'ordre les points de la liste.

```
>>> longueur([ori, p1, ori, p5])
5.0
>>> longueur([])
0.0
>>> longueur([p1, p2]) == distance(p1, p2)
True
```

Groupe	Nom	Prénom



Question 1.3 : [2/45]

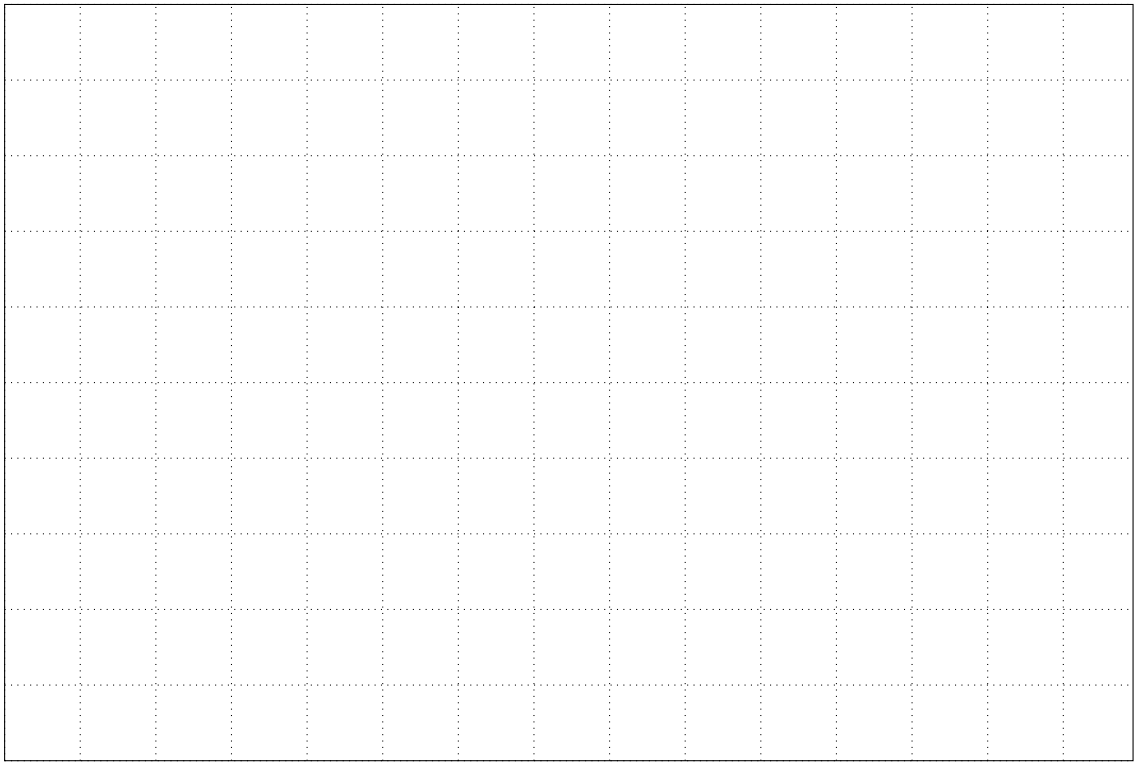
Le translaté d'un point (x, y) par le vecteur $((x_1, y_1), (x_2, y_2))$ est le point de coordonnées :

$$(x + x_2 - x_1, y + y_2 - y_1)$$

Ecrire une fonction `translate`, qui prend en entrée un point du plan `p` et un vecteur `v`, et qui renvoie le point du plan correspondant à `p` translaté de `v`.

```
>>> translate(ori, v3) == p4
True
>>> translate(ori, (p3, p3)) == ori
True
>>> translate(p4, v1)
(5.5, 2.0)
```

Groupe	Nom	Prénom



Question 1.4 : [3/45]

Ecrire une fonction `liste_translatee` qui prend en entrée une liste de points `li` et un vecteur `v` et qui renvoie la liste correspondant à `li` dans laquelle tous les points de `li` ont été translatés de `v`

```
>>> liste_translatee([p1, p2, p3], v1)
[(1.0, 3.0), (2.0, 5.0), (0.5, 2.0)]
>>> liste_translatee([p1, p2, p3], nul) == [p1, p2, p3]
True
```

Groupe	Nom	Prénom

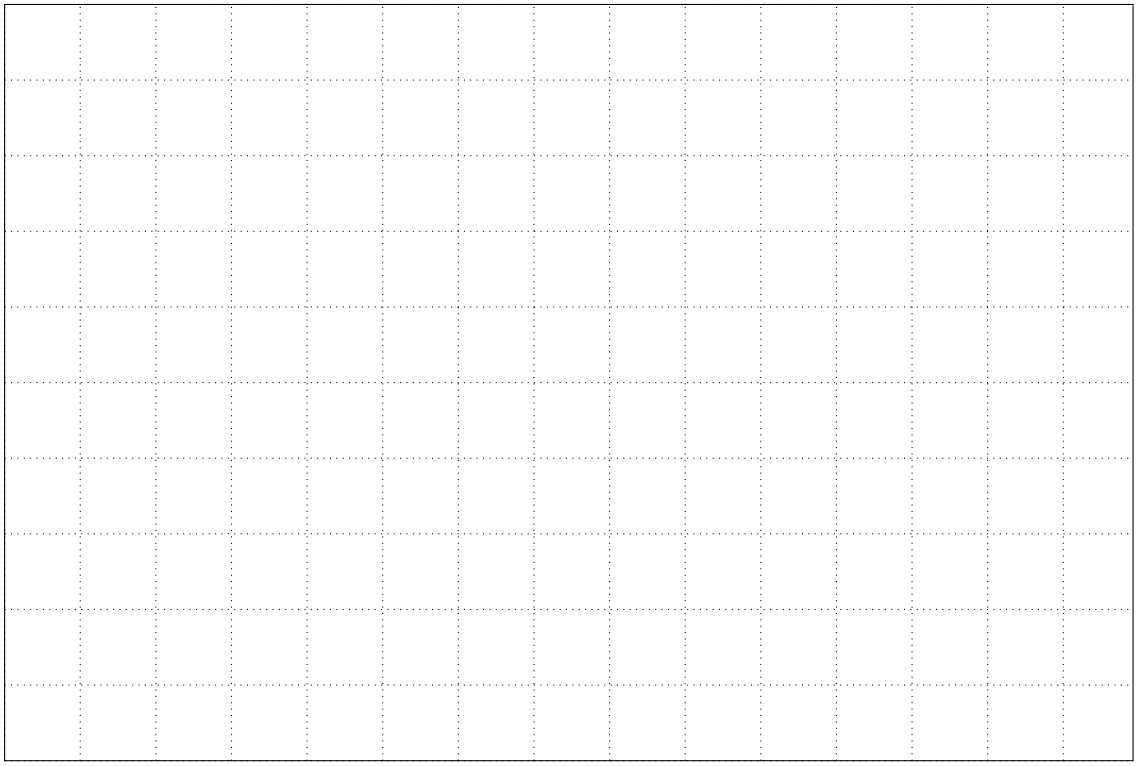
Question 1.5 : [1/45]

La *norme* du vecteur (p_1, p_2) est la distance entre ses deux extrémités, p_1 et p_2.

Ecrire une fonction `norme` qui prend en entrée un vecteur et calcule sa norme :

```
>>> norme((ori, ori))
0.0
>>> norme(v3)
4.5
>>> abs(norme(v1) - math.sqrt(5)) < 10 ** -12
True
```

Groupe	Nom	Prénom



Question 1.6 : [3/45]

Ecrire une fonction `non_nuls` qui prend en entrée une liste de vecteurs `li` et qui renvoie une liste correspondant à `li` dans laquelle tous les vecteurs dont la norme est nulle sont retirés.

```
>>> non_nuls([v1, v2, v3]) == [v1, v2, v3]
True
>>> non_nuls([v2, nul, v1, (p1, p1)]) == [v2, v1]
True
```

Groupe	Nom	Prénom

Groupe	Nom	Prénom

Exercice 2 : Codage de Gray

Dans cet exercice, nous considérons des chaînes de caractères composées uniquement des caractères '0' et '1'. On les appelle **chaînes de caractères binaires**. On dit que le caractère '0' est l'inverse du caractère '1', et réciproquement.

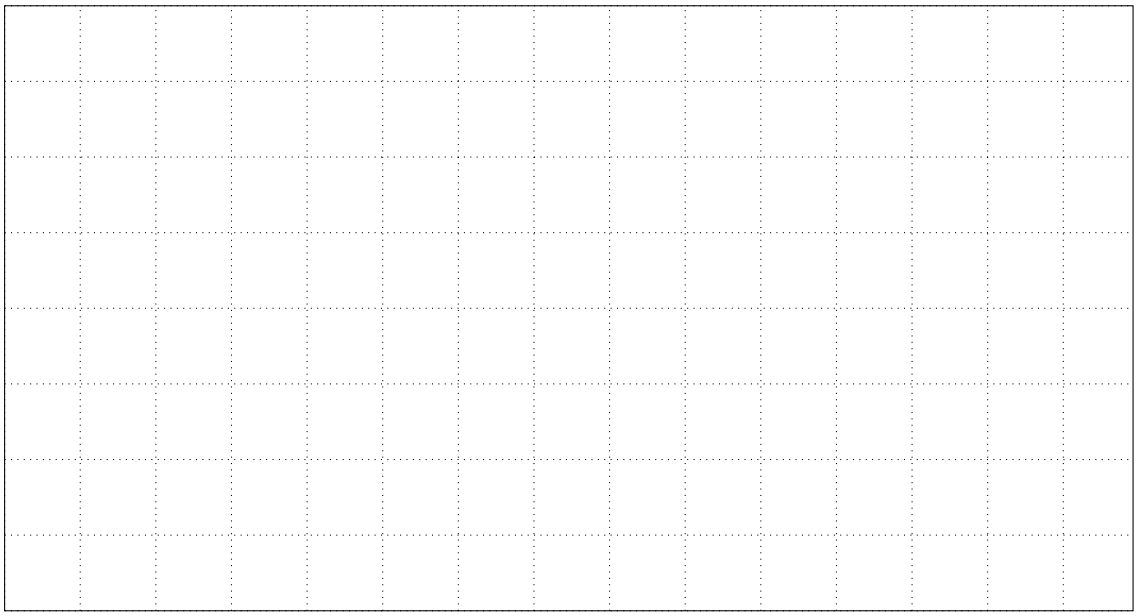
Suite de chaînes de Gray

Question 2.1 : [1/45]

Donner une définition de la fonction `inverse_char` qui, étant donné un caractère `c` valant '0' ou '1', renvoie le caractère représentant l'inverse de `c`.

Par exemple :

```
>>> inverse_char('0')
'1'
>>> inverse_char('1')
'0'
```



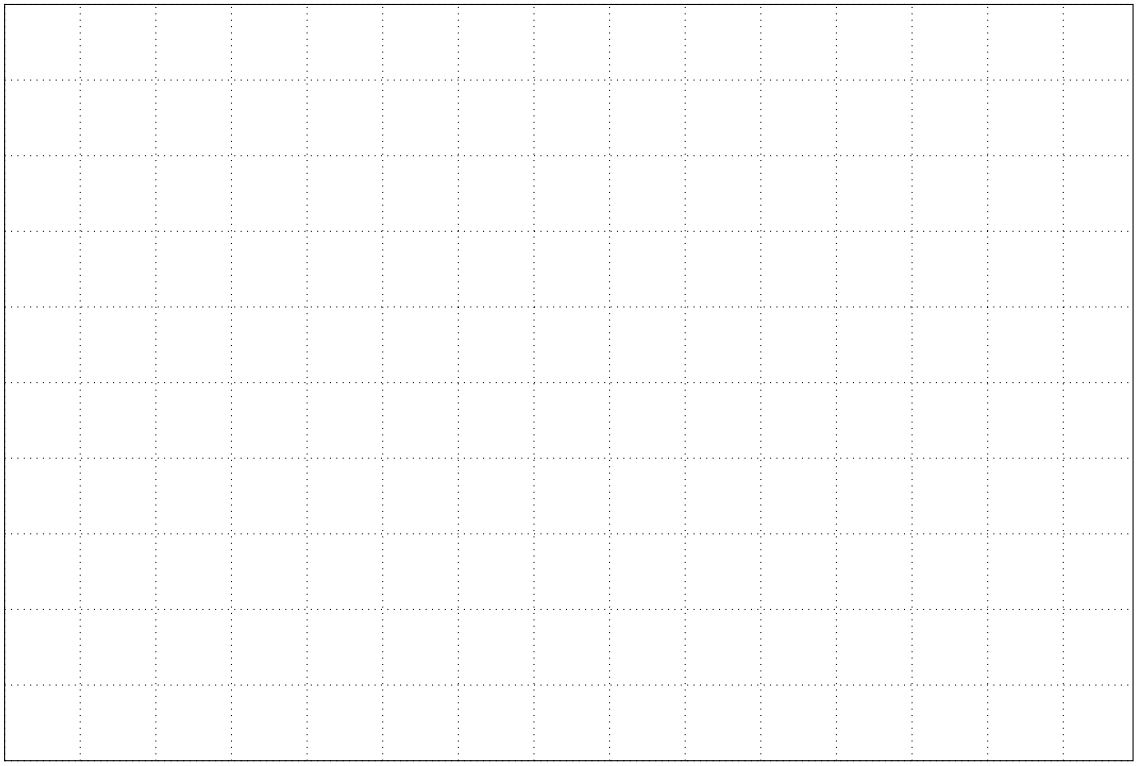
Question 2.2 : [2/45]

Donner une définition de la fonction `nb_un` qui, étant donné `s` une chaîne de caractères, renvoie le nombre d'occurrences du caractère '1' dans `s`.

Par exemple :

```
>>> nb_un('')
0
>>> nb_un('000')
0
>>> nb_un('001')
1
>>> nb_un('111')
3
>>> nb_un('abc')
0
```

Groupe	Nom	Prénom



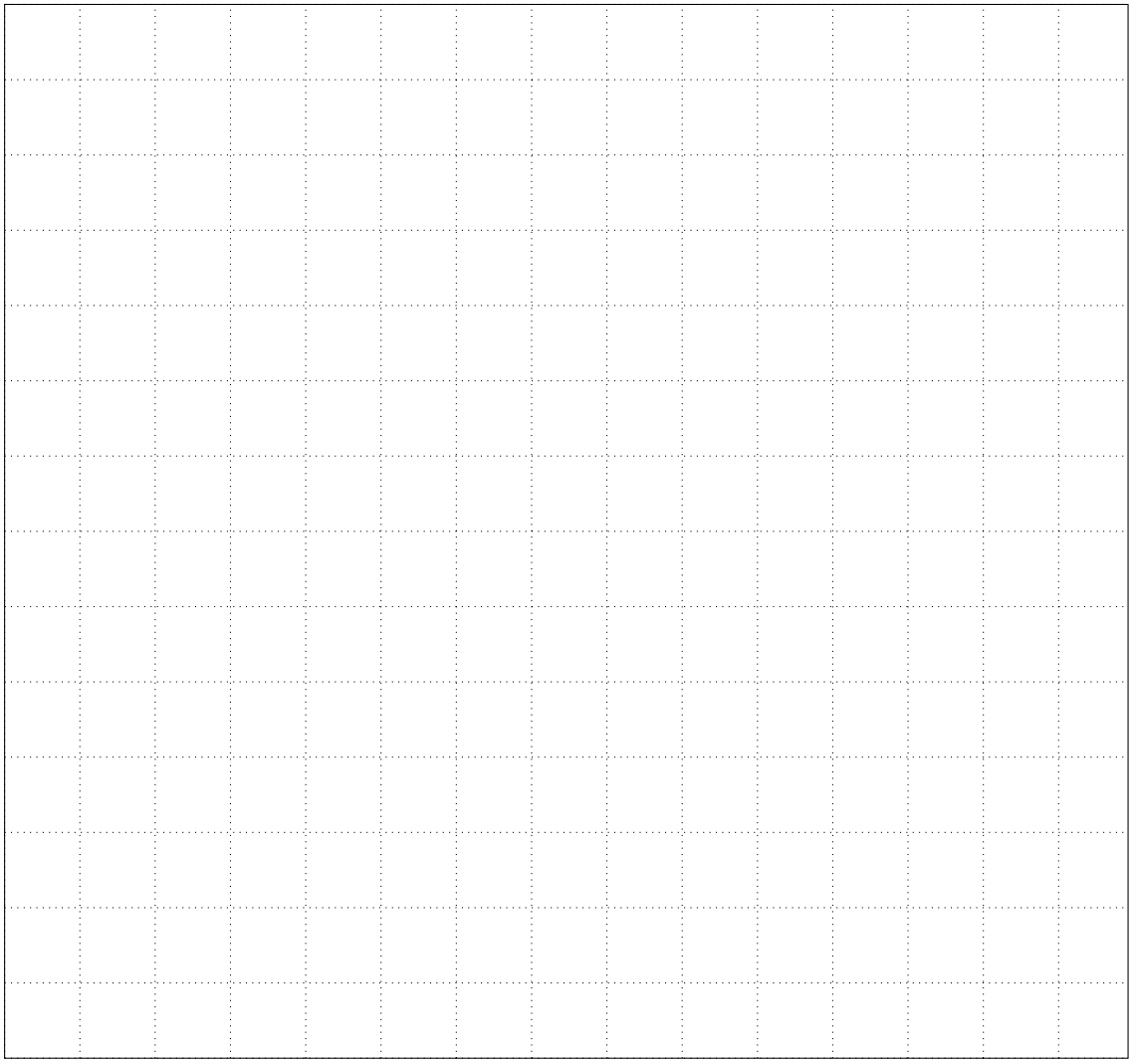
Question 2.3 : [4/45]

Donner une définition de la fonction `invc_avant_dernier_un` qui, étant donné `s` une chaîne de caractères binaires, renvoie la chaîne de caractères binaires `s` dans laquelle le caractère précédant l'occurrence la plus à droite de '1' a été inversé. Si cette inversion n'est pas possible, la chaîne renvoyée est égale à `s`.

Par exemple :

```
>>> invc_avant_dernier_un('')
''
>>> invc_avant_dernier_un('1')
'1'
>>> invc_avant_dernier_un('0')
'0'
>>> invc_avant_dernier_un('01')
'11'
>>> invc_avant_dernier_un('10')
'10'
>>> invc_avant_dernier_un('1001')
'1011'
>>> invc_avant_dernier_un('0011')
'0001'
>>> invc_avant_dernier_un('0000')
'0000'
>>> invc_avant_dernier_un('1000')
'1000'
```

Groupe	Nom	Prénom



Question 2.4 : [3/45]

Soit s une chaîne de caractères binaires non vide, on appelle *chaîne suivante de Gray* la chaîne de caractères binaires obtenue comme suit :

- Si le nombre d'occurrences de '1' dans s est pair alors la *chaîne suivante de Gray* de s est la chaîne s dans laquelle le dernier caractère a été inversé.
- Si le nombre d'occurrences de '1' dans s est impair alors la *chaîne suivante de Gray* de s est la chaîne de caractères s dans laquelle le caractère précédant l'occurrence de '1' la plus à droite a été inversé.

Donner une définition de la fonction `suivante_gray` qui, étant donné `s` une chaîne de caractères binaires non vide, renvoie la chaîne suivante de Gray de `s`.

Par exemple :

```
>>> suivante_gray('000')
'001'
>>> suivante_gray('001')
'011'
>>> suivante_gray('011')
```

Groupe	Nom	Prénom

```
'010'
>>> suivante_gray('010')
'110'
>>> suivante_gray('110')
'111'
>>> suivante_gray('111')
'101'
>>> suivante_gray('101')
'100'
>>> suivante_gray('100')
'100'
```

Question 2.5 : [4/45]

Donner une définition de la fonction `est_ordonnee` qui, étant donné `L` une liste de chaînes de caractères binaires non vides, teste si, dans `L`, toute chaîne `s` (sauf la dernière) est suivie de sa chaîne suivante de Gray.

Par exemple :

```
>>> est_ordonnee([])
True
>>> est_ordonnee(["1"])
True
>>> est_ordonnee(["0", "1"])
```

Groupe	Nom	Prénom

```
True
>>> est_ordonnee(['000', '001', '011', '010', '110', '111', '101', '100'])
True
>>> est_ordonnee(["000", "001", "010"])
False
```

Construction d'une suite

On peut construire L_n la liste contenant toutes les chaînes de caractères binaires de longueur $n > 1$ en procédant comme suit :

- Si $n = 1$ alors L_1 vaut ['0', '1']
- Si $n > 1$, L_n se construit à partir de L_{n-1} : elle est composée des éléments de L_{n-1} tous préfixés par '0' puis des éléments de L_{n-1} en ordre inverse et préfixés par '1'. Par exemple :
  L_2 = ['00', '01', '11', '10']
  L_3 = ['000', '001', '011', '010', '110', '111', '101', '100']

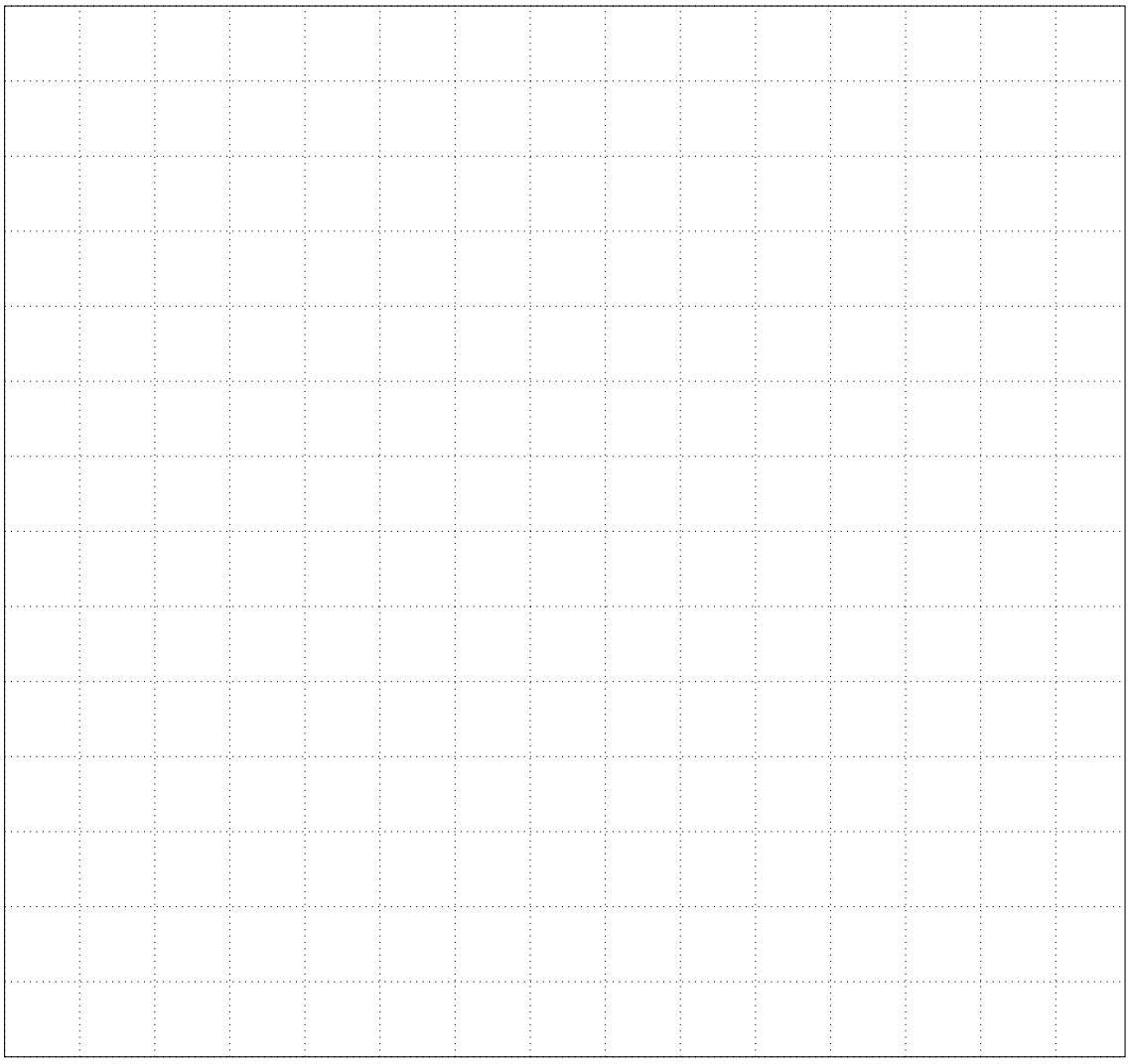
Groupe	Nom	Prénom

Question 2.6 : [3/45]

Donner une définition de la fonction `miroir_pref` qui, étant donné `L` une liste de chaînes de caractères, renvoie la liste composée des chaînes de `L` préfixées par '0' suivies des chaînes de `L` dans l'ordre inverse et préfixées par '1'.

Par exemple :

```
>>> miroir_pref([])
[]
>>> miroir_pref(["0", "1"])
['00', '01', '11', '10']
>>> miroir_pref(["a", "b"])
['0a', '0b', '1b', '1a']
```



Question 2.7 : [4/45]

Donner une définition de la fonction `liste_n` qui, étant donné `n` un entier naturel strictement positif, renvoie la liste comportant les éléments de L_n.

Par exemple :

Groupe | Nom | Prénom

```
>>> liste_n(1)
['0', '1']
>>> liste_n(2)
['00', '01', '11', '10']
>>> liste_n(3)
['000', '001', '011', '010', '110', '111', '101', '100']
```

Groupe	Nom	Prénom

Exercice 3 : Analyse de code

On considère la fonction `myst` dont les paramètres sont des entiers naturels :

```
def myst(a : [          ], b : [          ]) -> [          ]:
    """ Precondition : [                    ]
    """
    x : [          ]
    x = a
    y : [          ]
    y = b
    r : [          ]
    r = 0
    while x > 0:
        if x % 2 == 1:
            r = r + y
        x = x // 2
        y = y * 2
    return r
```

Question 3.1 : [1/45]

Compléter les informations de type et donner, si nécessaire, la précondition dans la définition de `myst` ci-dessus.

Question 3.2 : [3/45]

Compléter les quatre premières colonnes (colonnes **tour**, `r`, `x`, `y`) de la table de simulation **page 17** pour l'application : `myst(5,4)`.

Donner le résultat de l'application `myst(5,4)`.

Question 3.3 : [1/45]

Choisir laquelle de ces trois expressions booléennes est un invariant de boucle :

1. `a * b == x * y + r`
2. `a * b == x * y`
3. `a * b == a * b + r`

Groupe	Nom	Prénom

Question 3.4 : [1/45]

Vérifiez la validité de votre choix sur la table de simulation ci-dessous pour l'application `myst(5,4)`. Pour cela :

— recopiez l'égalité choisie sur la première ligne de la dernière colonne,

— pour les autres lignes de la dernière colonne, instanciez l'égalité (c'est-à-dire donnez l'égalité avec la valeur des éléments apparaissant dedans) et indiquez sa valeur entre parenthèses.

tour	`r`	`x`	`y`	
entrée				
1er				

Question 3.5 : [2/45]

Que calcule la fonction `myst` appliquée à deux paramètres `a` et `b` ? Expliquer comment on le déduit de l'invariant de boucle.

Groupe	Nom	Prénom

Question 3.6 : [2/45]

Quel est le résultat de `myst(30,10)` ? Combien de tours de boucle ont-ils été effectués ?

Mêmes questions pour `myst(32,10)` et `myst(1000,10)`