Partie A

1. (a) Une simple lecture du tableau « Ingredients » nous montre que les pizzas contenant de la mozzarella sont les pizzas ayant pour numéro 1, 2, 3, 8, 9 ou 11.

Pour savoir ce qu'affiche la procédure « Pizzas_avec », effectuons un tableau de suivi de variables :

	Nb_pizzas_valides	*Pizzas_valides*	*i*
Ligne 6	0	-	-
Ligne 7	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	-
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	0
Ligne 9	Test faux car Ingredients[0][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	1
Ligne 9	Test faux car Ingredients[1][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	2
Ligne 9	Test faux car Ingredients[2][0] vaut "roquette" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	3
Ligne 9	Test faux car Ingredients[3][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	4
Ligne 9	Test faux car Ingredients[4][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	5
Ligne 9	Test faux car Ingredients[5][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	6
Ligne 9	Test faux car Ingredients[6][0] vaut "tomate" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	7
Ligne 9	Test faux car Ingredients[7][0] vaut "jambon" → l.13		
Ligne 8	0	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	8
Ligne 9	Test vrai car Ingredients[8][0] vaut "mozzarella" → l.10		
Ligne 10	0	{8, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	8
Ligne 11	1	{99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	8
Ligne 8	1	{8, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	9
Ligne 9	Test vrai car Ingredients[9][0] vaut "mozzarella" → l.10		
Ligne 10	1	{8, 9, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	9
Ligne 11	2	{8, 9, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	9
Ligne 8	2	{8, 9, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	10
Ligne 9	Test faux car Ingredients[10][0] vaut "tomate" → l.13		
Ligne 8	2	{8, 9, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}	11
Ligne 9	Test vrai car Ingredients[11][0] vaut "mozzarella" → l.10		
Ligne 10	2	{8, 9, 11, 99, 99, 99, 99, 99, 99, 99, 99, 99}	11
Ligne 11	3	{8, 9, 11, 99, 99, 99, 99, 99, 99, 99, 99, 99}	11
Ligne 8	3	{8, 9, 11, 99, 99, 99, 99, 99, 99, 99, 99, 99}	12
	$i > 11$ donc on sort de la boucle → l.14		
FIN			

(b) Le problème est donc que l'algorithme ne teste que le premier ingrédient de chaque pizza, au lieu de tous les tester. On peut donc remplacer le test de la ligne 9 par un test de chacun des 4 ingrédients, ou bien rajouter une boucle imbriquée qui teste les 4 ingrédients.

Algorithme modifié avec un gros si.

Entrée :
Ingredient est une chaîne de caractères.

Variables :
Nb_pizzas_valides et i sont deux entiers.
Pizzas_valides est un tableau d'entiers.

Corps de l'algorithme :
```
1  Nb_pizzas_valides ← 0
2  Pizzas_valides ← {99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}
3  Pour i de 0 à 11, faire
4      Si (Ingredients[i][0] = Ingredient ou Ingredients[i][1] = Ingredient
        ou Ingredients[i][2] = Ingredient ou Ingredients[i][3] = Ingredient), alors
5          Pizzas_valides[Nb_pizzas_valides] ← i
6          Nb_pizzas_valides ← Nb_pizzas_valides + 1
7      Fin du Si
8  Fin du Pour
9  Retourner Pizzas_valides
```

Algorithme modifié avec une seconde boucle imbriquée.

Entrée :
Ingredient est une chaîne de caractères.

Variables :
Nb_pizzas_valides, i et j sont trois entiers.
Pizzas_valides est un tableau d'entiers.

Corps de l'algorithme :
```
1   Nb_pizzas_valides ← 0
2   Pizzas_valides ← {99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99}
3   Pour i de 0 à 11, faire
4       Pour j de 0 à 3, faire
5           Si Ingredients[i][j] = Ingredient, alors
6               Pizzas_valides[Nb_pizzas_valides] ← i
7               Nb_pizzas_valides ← Nb_pizzas_valides + 1
8           Fin du Si
9       Fin du Pour
10  Fin du Pour
11  Retourner Pizzas_valides
```

2. On remarque que cette question nécessite exactement le même raisonnement que dans le TP Loto, pour retrouver les numéros communs entre la grille jouée par l'humain et la grille générée aléatoirement par l'ordinateur !

   Appelons $t1$ et $t2$ les deux tableaux donnés en entrée. Il suffit, pour chaque élément e de $t1$, de tester l'appartenance ou pas de l'élément e au tableau $t2$. On peut donc utiliser une fonction auxiliaire « appartient » qui prend en entrée un élément e et un tableau t, et qui renvoie `Vrai` si $e \in t$, et `Faux` sinon. On peut aussi utiliser deux boucles imbriquées.

Fonction appartient.

Entrées :
e est un nombre entier.
t est un tableau d'entiers.

Variables :
i est un entier.

Corps de l'algorithme :
```
1  Pour i de 0 à longueur(t) − 1, faire
2      Si t[i] = e, alors
3          Retourner Vrai
4      Fin du Si
5  Fin du Pour
6  Retourner Faux
```

Fonction en_commun en utilisant la fonction appartient.

Entrées :
$t1$ et $t2$ sont deux tableaux d'entiers.

Variables :
i est un entier.
communs est un tableau d'entiers.

Corps de l'algorithme :
1 *communs* $\leftarrow$ { }
2 Pour i de 0 à $longueur(t1) - 1$, faire
3 Si $appartient(t1[i], t2)$, alors
4 $communs \leftarrow communs + \{t1[i]\}$
5 Fin du Si
6 Fin du Pour
7 Retourner *communs*

Fonction en_commun en utilisant deux boucles imbriquées.

Entrées :
$t1$ et $t2$ sont deux tableaux d'entiers.

Variables :
i et j sont deux entiers.
communs est un tableau d'entiers.

Corps de l'algorithme :
1 *communs* $\leftarrow$ { }
2 Pour i de 0 à $longueur(t1) - 1$, faire
3 Pour j de 0 à $longueur(t2) - 1$, faire
4 Si $t1[i] = t2[j]$, alors
5 $communs \leftarrow communs + \{t1[i]\}$
6 Fin du Si
7 Fin du Pour
8 Fin du Pour
9 Retourner *communs*

Remarque : les deux fonctions présentées ont le même comportement lorsque les tableaux donnés en entrée ne comportent pas de doublon (comme c'est le cas ici). Lorsque le premier tableau donné en entrée comporte des doublons, comparer les résultats des deux fonctions. De même si c'est le second tableau qui comporte des doublons. On pourra tester sur :

- $t1 = \{1, 3, 3, 5\}$ et $t2 = \{2, 3, 4, 5\}$
- $t1 = \{2, 3, 4, 5\}$ et $t2 = \{1, 3, 3, 5\}$

Partie B

1. Je donne ci-après le code Python de toutes les fonctions décrites plus haut, avec à chaque fois les deux manières proposées.

```
def Pizzas_avec_Modif1(Ingredient):
    Nb_pizzas_valides = 0
    Pizzas_valides = 12*[99]
    for i in range(12):
        if (Ingredients[i][0] == Ingredient or Ingredients[i][1] == Ingredient
         or Ingredients[i][2] == Ingredient or Ingredients[i][3] == Ingredient):
            Pizzas_valides[Nb_pizzas_valides] = i
            Nb_pizzas_valides = Nb_pizzas_valides + 1
    return Pizzas_valides
```

```
def Pizzas_avec_Modif2(Ingredient):
    Nb_pizzas_valides = 0
    Pizzas_valides = 12*[99]
    for i in range(12):
        for j in range(4):
            if (Ingredients[i][j] == Ingredient):
                Pizzas_valides[Nb_pizzas_valides] = i
                Nb_pizzas_valides = Nb_pizzas_valides + 1
    return Pizzas_valides

def Appartient(e, t):
    for i in range(len(t)):
        if t[i] == e:
            return True
    return False

def En_commun1(t1, t2):
    communs = []
    for i in range(len(t1)):
        if Appartient(t1[i], t2):
            communs = communs + [t1[i]]
    return communs

def En_commun2(t1, t2):
    communs = []
    for i in range(len(t1)):
        for j in range(len(t2)):
            if (t1[i] == t2[j]):
                communs = communs + [t1[i]]
    return communs
```

2. (a)
```
Ingredient = input("Entrez un nom d'ingrédient :")
contient_ingredient = Pizzas_avec_Modif1(Ingredient)
print("Les pizzas contenant l'ingrédient", Ingredient, "sont les pizzas suivantes :")
for i in range(12):
    if contient_ingredient[i] != 99:
        print(Noms[contient_ingredient[i]])
```

   (b)
```
Ingredient1 = input("Entrez un premier nom d'ingrédient :")
Ingredient2 = input("Entrez un second nom d'ingrédient :")
contient_ingredient1 = Pizzas_avec_Modif1(Ingredient1)
contient_ingredient2 = Pizzas_avec_Modif1(Ingredient2)
print("Les pizzas contenant à la fois l'ingrédient", Ingredient1, "et l'ingrédient",
  Ingredient2, "sont les pizzas suivantes :")
contient_les_deux = En_commun1(contient_ingredient1, contient_ingredient2)
for i in range(len(contient_les_deux)):
    if contient_les_deux[i] != 99:
        print(Noms[contient_les_deux[i]])
```