UNIVERSITÉ CÔTE D'AZUR

Seconde session de Bases de l'informatique 1

L'usage ou la possession de tout appareil électronique sera considéré comme de la fraude. Documents interdits.

La notation est donnée à titre indicatif.

DATE : 15/06/2026 DURÉE : 2 heures NOTE : ..... /20

NOM : .....

PRÉNOM : .....

NUMÉRO D'ÉTUDIANT : .....

Exercice 1 : Calculs binaires (3,5 points)

1. En utilisant l'algorithme vu en cours, écrire 113 en binaire.

```
113 = 2 × 56 + 1
 56 = 2 × 28 + 0 # J'enleve des points  pour ceux
 28 = 2 × 14 + 0 # utilisant une autre présentation que :
 14 = 2 ×  7 + 0 # … = 2 × … + …
  7 = 2 ×  3 + 1
  3 = 2 ×  1 + 1
  1 = 2 ×  0 + 1
113 est donc représenté par 111 0001 en binaire
```

2. En déduire la représentation binaire signée sur 8 bits de -113.

```
On part de 113 sur 8 bits   : 0111 0001
On prend le complément à 2  : 1000 1110
et on ajoute 1              : 1000 1111

-113 est donc représenté par 1000 1111  en binaire sur 8 bits
```

3. En utilisant l'algorithme vu en cours, donnez l'écriture décimale du nombre positif dont la représentation binaire est 101 1101.

```
Sur 8 bits 0101 1101 commence par 0, c'est un nombre positif
1        = 2 × 0 + 1 = 1
10       = 2 × 1 + 0 = 2
101      = 2 × 2 + 1 = 5
101 1    = 2 × 5 + 1 = 11
101 11   = 2 × 11 + 1 = 23
101 110  = 2 × 23 + 0 = 46
101 1101 = 2 × 46 + 1 = 93

101 1101 vaut donc 93
```

4. Donnez l'écriture décimale du nombre dont la représentation signée sur 8 bits binaire est `1010 0001`.

```
Le nombre commençant par 1 sur 8 bits, c'est un négatif
On a la représentation sur 8 bits : 1010 0001
On prend le complément à 2        : 0101 1110
et on ajoute 1                    : 0101 1111
On reconnaît 93 + 2 = 95, donc le nombre de départ est -95.
```

Exercice 2 : Dire bonjour : « Saluton ! » (4,5 points)

L'objectif de cet exercice et du suivant est de partir d'un corpus de texte (une succession de phrases comme ci-dessous) et d'y analyser la langue. On supposera que les chaînes ne sont constituées que de lettres et d'espaces (ni chiffres, ni ponctuations, ni autres symboles bizarres).

```
1
2  liste_phrases = [ "Bonjour camarade",
3                    "Saluton",
4                    "Saluton al vi",
5                    "Ciao bella pasta",
6                    "Bonjour",
7                    "Bonjour toi",
8                    "Bung"
9                    "Yeah america",
10                   "Wesh",
11                   "Wesh Wesh les amis",
12                   "Saluton amiko mia" ]
13
```

1. Pour cela nous allons extraire le premier mot de chaque phrase. Écrire une fonction `premier_mot(chaîne)` qui renvoie le premier mot de la chaîne. Si la chaîne est vide ou commence par un espace, on renverra la chaîne vide. Il sera **interdit** d'utiliser la méthode `split` ou les *slices* (`chaîne[a:b:c]`).

```
1  >>> premier_mot("Saluton al vi")
2  'Saluton'
3  >>> premier_mot("Wesh")
4  'Wesh'
5  >>> premier_mot(" Bonjour espace")
6  ''
7  >>> premier_mot("")
8  ''
```

```
def premier_mot(chaîne):
    rés = ""
    for c in chaîne:
        if c == " ":
            return rés
        else:
            rés = rés + c
    return rés
```

2. On souhaite stocker ces mots dans une liste sans duplications. Écrire la fonction `ajout_si_nouveau(chaîne, liste)` qui ajoute, si elle n'y est pas déjà, la chaîne en fin de liste. La liste en paramètre sera **modifiée**. On ne pourra pas tester l'appartenance d'un mot à une liste avec le mot clé `in` ; il faudra parcourir la liste donnée en paramêtre avec une boucle.

```
>>> L
['Saluton', 'Ciao']
>>> ajout_si_nouveau("Bonjour", L) # Bonjour n'étant pas dans L sera ajouté à la fin
>>> L
['Saluton', 'Ciao', 'Bonjour']
>>> ajout_si_nouveau("Ciao", L) # Ciao ne sera pas ajoutée.
>>> L
['Saluton', 'Ciao', 'Bonjour']
```

```
def ajout_si_nouveau(chaîne, liste):
    for ch in liste:
        if ch==chaîne:
            return
    liste.append(chaîne)
```

3. En utilisant ces deux dernières fonctions, déduire `liste_mots_uniques(liste_phrases)` qui renvoie la liste (sans répétition) de tous les premiers mots des phrases de la liste en paramètre.

```
>>> liste_mots_uniques(liste_phrases)
['Bonjour', 'Saluton', 'Ciao', 'Yeah', 'Wesh']
```

```
def liste_mots_uniques(liste):
    rés = []
    for chaîne in liste:
        mot = premier_mot(chaîne)
        ajout_si_nouveau(mot, rés)
    return rés
```

4. On se rappelle soudain l'existence en Python d'un type ensemble permettant de gérer des collections sans duplications. Écrire la fonction `ensemble_mots(liste)` qui construit et renvoit un ensemble contenant tous les premiers mots des phrases de la liste en paramètre. Votre fonction devra construire directement l'ensemble sans utiliser les fonctions précédentes ni une liste intermédiaire.

```
>>> ensemble_mots(liste_phrases)
{'Bonjour', 'Ciao', 'Yeah', 'Wesh', 'Saluton'}
```

```
def ensemble_mots(liste):
    rés = set()
    for chaîne in liste:
        mot = premier_mot(chaîne)
        rés.add(mot)
    return rés
```

Exercice 3 : Statistiques linguistique (6,5 points)

Maintenant que nous avons une liste de mots (voir exercice précédent), nous pouvons les analyser. Pour cela nous utilisons un dictionnaire associant à chaque langue une liste de mots.

```
dico_bonjour = { "Fr" : ["Bonjour", "Salut"] ,     # Français
                 "Us" : ["Hello", "Hi", 'Yeah'] , # Américain
                 "Eo" : ["Saluton"],              # Espérantiste
                 "Jp" : ["Wesh"],                 # Jeunesse périurbaine
                 "It" : ["Ciao", "Buongiorno"] }  # Italien
```

1. Nous souhaitons créer un dictionnaire inverse du précédent dans lequel les clés sont les mots et les valeurs associées sont les langues. Écrire la fonction `inverser_dico(dico)` construisant une telle inversion en renvoyant le dictionnaire inverse.

```
>>> dico_bonjour["Fr"]
['Bonjour', 'Salut']
>>> dicinv = inverser_dico(dico_bonjour)
>>> dicinv["Salut"]
'Fr'
>>> dicinv["Bonjour"]
'Fr'
```

```
def inverser_dico(dico):
    d = dict()
    for langue in dico:
        for mot in dico[langue]:
            d[mot] = langue
    return d
```

2. On souhaite calculer à partir d'un dictionnaire inversé et d'une liste de mots, le nombre d'occurrences associé à une langue donnée en paramètre. Écrire `nombre_occurrences(langue, liste_mots, dicinv)` qui compte et renvoie le nombre de mots de la langue en paramètre.

```
>>> liste_mots
['Bonjour', 'Saluton', 'Yeah', 'Wesh', 'Saluton', 'Ciao', 'Bonjour', 'Salut', 'Saluton']
>>> nombre_occurrences("Fr", liste_mots, dicinv) # 'Bonjour', 'Bonjour' et 'Salut'
3
>>> nombre_occurrences("Es", liste_mots, dicinv) # Et les mots espagnols ?
0
>>> nombre_occurrences("It", liste_mots, dicinv) # 'Ciao'
1
```

```
def nombre_occurrences(langue, liste_mots, dicinv):
    n = 0
    for mot in liste_mots:
        if dicinv[mot] == langue:
            n = n+1
    return n
```

3. En une seule ligne et en utilisant une *compréhension*, définir une liste `langues` contenant toutes les clés du dictionnaire `dico_bonjour`.

```
langues = [ k for k in dico_bonjour ]
```

4. On souhaite calculer pour les mots de la liste, le nombre d'occurrences pour chacune des langues. Écrire la fonction `statistique(liste_mots, dico_langues)` qui renverra une liste de tuples de la forme `(Langue, n)`. Par exemple dans la liste de mots de la question 2, il y a trois mots français, un mot américain, trois mots espéranto, etc.

```
>>> statistique(liste_mots, dico_bonjour)
[('Fr', 3), ('Us', 1), ('Eo', 3), ('Jp', 1), ('It', 1)]
```

```
def statistique(liste_mots, dico_langues):
    dico = inverser_dico(dico_langues)
    res = []
    for langue in dico_langues:
        n = nombre_occurrences(langue, liste_mots, dico)
        res.append((langue, n))
    return res
```

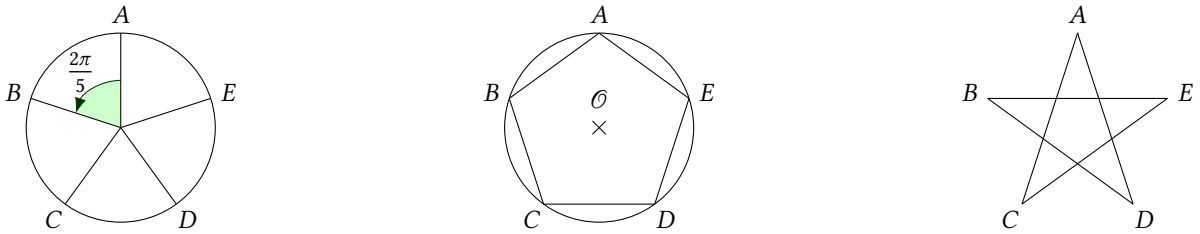
5. On souhaite analyser une telle liste de couples pour savoir si l'Espéranto s'est bien imposé comme langue internationale (on parle alors de victoire finale, *fina venko*). Écrire la fonction `fina_venko(stat)` qui renvoie `True` si l'Espéranto est la seule langue la plus utilisée et `False` sinon. Pour avoir tous les points, il ne faut faire qu'un passage dans la liste avec une boucle.

```
>>> stat_mots_français  = [('Fr', 4), ('Us', 1), ('Eo', 3), ('Jp', 1), ('It', 1)]
>>> stat_mots_égalité   = [('Fr', 3), ('Us', 1), ('Eo', 3), ('Jp', 1), ('It', 1)]
>>> stat_mots_espéranto = [('Fr', 3), ('Us', 1), ('Eo', 4), ('Jp', 1), ('It', 1)]
>>> fina_venko(stat_mots_français)  # C'est le Français qui gagne
False
>>> fina_venko(stat_mots_égalité)   # Égalité
False
>>> fina_venko(stat_mots_espéranto) # C'est l'Espéranto qui gagne seul !
True
```

```
def fina_venko(stat):
    langue_max = None
    n_max = 0
    for couple in stat:
        (langue, n) = couple
        if n>n_max:
            langue_max = langue
            n_max = n
        elif n==n_max:
            langue_max = None
    return "Eo" == langue_max
```

Exercice 4 : L'étoile espérantiste : « La stelo » (5,5 points)

On souhaite dans cet exercice tracer le symbole de l'Espéranto : une étoile à cinq branches. Pour cela, nous allons partir d'un point A de coordonnées (0,1) sur le cercle unité. Par rotation successive d'angle $\frac{2\pi}{5}$ on obtient les points du pentagone A, B, C, D, et E. Il suffit alors de les relier dans le bon ordre.



1. Commencez pas créer une classe `Point` avec deux attributs `x` et `y`. La classe pourra s'utiliser comme dans l'exemple ci-dessous.

```
>>> p = Point(5,2)
>>> (p.x, p.y)
(5, 2)
```

```
class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
```

2. Il y a deux systèmes de coordonnées, le système mathématique avec le point $\mathcal{O}$ au centre et le système informatique (utilisé en Tk). Expliquer la différence entre les deux systèmes. On supposera jusqu'à la fin de l'exercice que l'on travaille dans le système de coordonnées mathématique.

```
1. En informatique le point de coordonnées (0,0) est dans le coin en haut à gauche de la fenêtre.

2. En informatique les ordonnées (y) sont orientées vers le bas et non vers le haut
```

3. Écrire la fonction `rotation(p,alpha)`, où `p` est un objet de la classe `Point`, qui calcule et renvoie le point image de la rotation du point `p` de centre $\mathcal{O}$ et d'angle `alpha`. On rappelle que le résultat d'une rotation d'angle α d'un point $p = (x_0, y_0)$, autour du point origine du repère $\mathcal{O}(0,0)$) donne un point p_α de coordonnées :

$$\begin{cases} x = x_0 \cos(\alpha) - y_0 \sin(\alpha) \\ y = x_0 \sin(\alpha) + y_0 \cos(\alpha) \end{cases}$$

```
# On suppose que l'import suivant a été fait préalablement.
from math import cos, sin, pi
```

```
def rotation(p,alpha):
        new_x = p.x * cos(a) - p.y * sin(a)
        new_y = p.x * sin(a) + p.y * cos(a)
        return Point(new_x, new_y)
```

4. Avec une boucle `for` et à l'aide de la fonction précédente, écrire la fonction `pentagone(A)` qui à partir d'un point initial, renvoie la liste de tous les points du pentagone, centré sur $\mathcal{O}(0,0)$ et dont le premier sommet est A.

```
def pentagone(p):
    res = []
    a = 2*pi/5
    for x in range(5):
        res.append(p)
        p = rotation(p,a)
    return res
```

5. En utilisant la fonction précédente, écrire une fonction `étoile(p)` qui renvoie la liste des points du pentagone mais dans l'ordre nécessaire pour tracer l'étoile (c'est-à-dire A, C, E, B et D).

```
def étoile(p):
    liste = pentagone(p)
    indices = [0,2,4,1,3]
    res = []
    for i in indices:
        res.append(liste[i])
    return res
```

6. On suppose fournie une fonction `ligne(p,q)` qui relie par un segment les deux points. En utilisant cette fonction, écrire la fonction `stelo()` qui trace l'étoile espérantiste ! Quelle belle conclusion pour cette magnifique épreuve.

```
def stelo():
    A = Point(0,1)
    liste = étoile(A)
    for i in range(len(liste)-1):
        ligne(liste[i], liste[i+1])
    ligne(liste[-1], liste[0]
```