UNIVERSITÉ CÔTE D'AZUR

# Partiel 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 : 04/11/2026 DURÉE : 2 heures NOTE : ..... /20

NOM : .....

PRÉNOM : .....

NUMÉRO D'ÉTUDIANT : .....

*Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire ; chaque question est donc indépendante.*

*À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncer vous conseille l'utilisation de certaines d'entre elles.*

```
# Fonctions et boucles autorisées
range(…) len(…) print(…)
Boucles while et boucles for avec un range

# Déstructuration autorisée
(x,y) = p # où p est un tuple de longueur 2
```

```
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(…) min(…) sum(…) abs(…) L.append(…) s.split(…) s.index(…) L.extend(…)

# Les boucles itérant directement sur les séquences sont interdites
for x in L:

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices
# À la place vous devez utiliser des boucles.
x in L
[ x for x in range(L) ]
chaîne[début:fin:pas]

# Les tableaux devront être créés à la bonne taille puis modifiés.
# Interdiction d'agrandir un tableau avec append ou la concaténation
L = L + [3]
L.append(3)
```

# Exercice 1 : Lire du code (2 points)

1. Qu'affiche l'exécution de `saucisse(4)`.

```python
def saucisse(c):
    a = 1
    b = 2
    for i in range(c):
        print(a)
        if i%2==0:
            a = a+b
            b = a
        else:
            a = a//b
            b = a
```

```
1
3
1
2
```

2. Qu'affiche l'exécution de `choucroute("10010")`. On indiquera aussi la valeur de retour.

```python
def choucroute(c):
    t = [0, 2, 4, 0, 2, 4]
    d = 0
    for i in range(len(c)):
        if c[i] == "0":
            d = t[0+d]
        else:
            d = t[1+d]
        print(">>>", d)
    return d//2

```

```
>>> 2
>>> 4
>>> 2
>>> 0
>>> 0
0
```

## Exercice 2 : Calculs binaires (4 points)

1. En utilisant l'algorithme vu en cours, écrire 109 en binaire.

```
109 = 2 × 54 + 1
 54 = 2 × 27 + 0 # J'enlève des points  pour ceux
 27 = 2 × 13 + 1 # utilisant une autre présentation que :
 13 = 2 ×  6 + 1 # … = 2 × … + …
  6 = 2 ×  3 + 0
  3 = 2 ×  1 + 1
  1 = 2 ×  0 + 1
109 est donc représenté par 110 1101 en binaire
```

2. En déduire la représentation binaire signée sur 8 bits de -109.

```
On part de 109 sur 8 bits   : 0110 1101
On prend le complément à 2  : 1001 0010
et on ajoute 1              : 1001 0011

-109 est donc représenté par 1001 0011  en binaire sur 8 bits
```

3. En utilisant l'algorithme du cours, donnez la valeur du nombre positif dont la représentation binaire est `111 0001`.

```
Sur 8 bits 0111 0001 commence par 0, c'est un nombre positif
1        = 2 × 0 + 1 = 1
11       = 2 × 1 + 1 = 3
111      = 2 × 3 + 1 = 7
111 0    = 2 × 7 + 0 = 14
111 00   = 2 × 14 + 0 = 28
111 000  = 2 × 28 + 0 = 56
111 0001 = 2 × 56 + 1 = 113

111 0001 vaut donc 113
```

4. Donnez la valeur du nombre dont la représentation signée sur 8 bits binaire est `1001 0000`.

```
Le nombre commençant par 1 sur 8 bits, c'est un négatif
On a la représentation sur 8 bits : 1001 0000
On prend le complément à 2        : 0110 1111
et on ajoute 1                    : 0111 0000
On reconnaît 112 = 113 - 1, donc le nombre de départ est -112.
```

## Exercice 3 : Recherche dichotomique de racines (4 points)

1. Écrire une fonction `f(x)` correspondant au polynôme $f : x \to x^4 - 17x^3 - 655$.

```
def f(x):
    return x**4 - 17*x**3 - 655
```

2. Écrire un test avec `assert` pour montrer que $f$ est strictement négatif en 1.

```
assert f(1) < 0
```

3. La fonction $f$ tend vers l'infini lorsque $x$ tend vers l'infini. On peut donc trouver un $x$ suffisamment grand tel que $f(x)$ soit positif. Écrire une fonction `trouver_valeur_positive(f)` qui renvoie $x$ tel que $f(x)$ soit positif. Votre code doit fonctionner quelle que soit la fonction `f`, pourvu qu'elle tende vers l'infini. On commencera à tester avec $x = 1$ puis on doublera la valeur de $x$ à chaque itération jusqu'à trouver une valeur rendant $f$ positive ou nulle.

```
def trouver_valeur_positive(f):
    x = 1
    while f(x) < 0:
        x = 2*x
    return x
```

4. Écrire la fonction `presque_nulle(y)` qui renvoie `True` si y est nulle à $10^{-3}$ près et `False` sinon. La fonction devra tenir sur une ligne et on aura le droit exceptionnellement d'utiliser la fonction `abs` (inutile de me remercier).

```
def presque_nulle(y):
    return abs(y) < 0.001 # ou abs(y) <= 0.001
```

5. Ayant maintenant une valeur négative (question 2) et une valeur positive (question 3), cela signifie qu'il y a entre ces deux bornes une racine $x_0$ tel que $f(x_0) = 0$ (théorème des valeurs intermédiaires).

Écrire une fonction `trouver_racine(f,a,b)` qui cherche et renvoie une racine $x_0$ dans l'intervalle $[a, b]$. On suppose que les bornes de l'intervalle vérifient bien la propriété d'encadrement « $f(a) \leq 0 \leq f(b)$ ». On commencera par calculer le milieu `m` de $[a, b]$. Selon le signe de `f(m)` on recommence le calcul sur l'intervalle $[m, b]$ ou $[a, m]$ afin de conserver la propriété d'encadrement. On s'arrête lorsque `f(m)` est presque nulle. On pourra faire une boucle ou une récursion.

```
def trouver_racine(f, a, b):
    m = (a+b)/2
    y = f(m)
    while not presque_nulle(y):
        if y < 0:
            a = m
        else:
            b = m
        m = (a+b)/2
        y = f(m)
    return m
```

## Exercice 4 : Encodage de l'unicode (5 points)

Nous avons vu en cours que chaque caractère se voit associer un numéro unicode : jusqu'à 127 pour l'ASCII et au delà pour le reste. Le codage UTF-8 d'unicode représente les caractères sur 1, 2, 3 ou 4 octets selon leur numéro.

- Les caractères au numéro strictement inférieur à $n_1 = 80_{16}$ sont codés sur un octet (ASCII, exemples : `123abc`)
- ceux strictement inférieurs à $n_2 = 800_{16}$ sont représentés sur deux octets. (correspondant aux accents : `àéüçœ`)
- ceux strictement inférieurs à $n_3 = 10000_{16}$ sont représentés sur trois octets. (écritures asiatiques : `ワンピース`)
- les autres sur 4 octets. (autres : 😀😉😕)

1. Les valeurs $n_1$, $n_2$ et $n_3$ ont été décrites en notation hexadécimale (base 16). Définissez ces variables en Python en calculant leurs valeurs à partir de la base 16 . On ne pourra évidemment pas utiliser la notation `n1=0x80`, valide en Python.

```
n1 = 8 * 16
n2 = 8 * 16 * 16 # ou 8 * 16**2
n3 = 16 ** 4
```

2. Écrire une fonction `taille(c)` qui renvoie la taille d'un caractère en octets.

```
>>> taille("A")  # ASCII
1
>>> taille("à")  # Français
2
```

```
>>> taille("ワ")  # Japonais
3
>>> taille("😀") # Émoji
4
```

```
def taille(c):
    numéro = ord(c)
    if numéro < n1:
        return 1
    elif numéro < n2:
        return 2
    elif numéro < n3:
        return 3
    else:
        return 4
```

3. Écrire une fonction `longueur_en_octets(chaîne)` renvoyant la longueur de la chaîne, longueur obtenue en sommant la taille de chacun de ses caractères.

```
>>> longueur_en_octets("C😉😀1") # 1 + 4 + 4 + 1
10
```

```
def longueur_en_octets(chaîne):
    n = 0
    for i in range(len(chaîne)):
        n = n + taille(chaîne[i])
    return n
```

4. Écrire une fonction `pure_ascii(chaîne)` qui renvoie `True` si la chaîne ne contient que des caractères codés sur un octet et `False` sinon. La fonction devra parcourir la chaîne avec une boucle et s'interrompre dès que possible.

```
def pure_ascii(chaîne):
    for i in range(len(chaîne)):
        if taille(chaîne[i]) > 1:
            return False
    return True
```

5. Écrire une fonction `plus_longue_chaîne(tableau)` qui renvoie la plus longue chaîne (en octets) du tableau donné en paramètre. En cas d'égalité, on renverra la première chaîne maximisant la longueur.

```
>>> plus_longue_chaîne(["oui", "été", "😀", "😀B"]) # les longueurs sont 3, 5, 4, et 5
'été'
```

```
def plus_longue_chaîne(tableau):
    nmax = 0
    smax = ""
    for i in range(len(tableau)):
        n = longueur_en_octets(tableau[i])
        if n>nmax:
            nmax = n
            smax = tableau[i]
    return smax
```

6. Écrire `plus_longs_caractères(chaîne)` qui renvoie la chaîne contenant tous les caractères de taille maximale.

```
>>> plus_longs_caractères("abcéçũ")
'éçũ'
>>> plus_longs_caractères("abcワンピースéçũ")
'ワンピース'
```

```
>>> plus_longs_caractères("123éà😀ũ😐")
'😀😐'
>>> plus_longs_caractères("chaîne")
'î'
```

```
def les_plus_longs_caractères(chaîne):
    nmax = 0
    smax = ""
    for i in range(len(chaîne)):
        n = taille(chaîne[i])
        if n==nmax:
            nmax = n
            smax = smax + chaîne[i]
        elif n>nmax:
            nmax= n
            smax = chaîne[i]
    return smax
```

## Exercice 5 : Calcul de diff (5 points)

1. Écrire une fonction `tableau_vers_chaîne(tableau)` qui renvoie la chaîne correspondant à la concaténation de ses éléments.

```
>>> tableau_vers_chaîne(["E", "s", "p", "e", "r", "a", "n", "t", "o"])
'Esperanto'
```

```
def tableau_vers_chaîne(tab):
    ch = ""
    for i in range(len(tab)):
        ch = ch + tab[i]
    return ch
```

2. On se donne deux chaînes de même longueur et on cherche à indiquer les modifications à faire pour passer de la première à la seconde. Concrètement, pour passer de `"Olivier"` à `"Voilier"`, on peut conserver les lettres `"--i-ier"` et il faut remplacer `"Ol-v---"` par `"vo-l---"`.

Écrire la fonction `calculer_diff(ch1,ch2)` qui renvoie le tableau qui pour chaque indice contient `None` si les deux caractères à cet indice sont égaux et le caractère de ch2 sinon. Pour passer de `"Olivier"` à `"voilier"`, il faut remplacer le premier caractère par `"v"`, le second par `"o"`, ne pas toucher au troisième, remplacer le quatrième par `"l"` et ne pas toucher au reste.

```
>>> calculer_diff("Olivier", "voilier")
['v', 'o', None, 'l', None, None, None]
>>> calculer_diff("aaaaa", "aaBaa")
[None, None, 'B', None, None]
```

```
def calculer_diff(ch1,ch2):
    tab = [ None ] * len(ch1)
    for i in range(len(ch1)):
        if ch1[i] == ch2[i]:
            tab[i] = None
        else:
            tab[i] = ch2[i]
    return tab
```

3. On cherche, à partir du tableau de différences renvoyé dans la question précédente, à calculer le nombre de lettres à modifier. Écrire `nombre_différences(tab)` qui renvoie le nombre de cases du tableau différentes de `None`.

```
>>> nombre_différences(['v', 'o', None, 'l', None, None, None])
3
>>> nombre_différences([ None, None, "B", None, None])
1
```

```
def nombre_différences(tab):
    n = 0
    for i in range(len(tab)):
        if tab[i] != None:
            n=n+1
    return n
```

4. Si les deux chaînes sont proches, le tableau de différences calculé précédemment est inutilement grand. On souhaite se débarrasser des `None` en ne conservant que la nouvelle lettre ainsi que son indice. Écrire la fonction `compresser(tab)` qui à partir d'un tableau de différences renvoie un tableau de couples de la forme `(indice, caractère)`.

```
>>> compresser(['v', 'o', None, 'l', None, None, None])
[(0, 'v'), (1, 'o'), (3, 'l')]
>>> compresser([ None, None, "B", None, None])
[(2, 'B')]
```

```
def compresser(tab):
    n = 0
    t = [ None ] * nombre_différences(tab)
    j = 0
    for i in range(len(tab)):
        if tab[i] != None:
            t[j] = (i, tab[i])
            j = j+1
    return t
```

On peut ainsi définir la fonction `diff(ch1, ch2)` qui calcule le `diff` entre deux chaînes.

```
def diff(ch1, ch2):
    return compresser(calculer_diff(ch1, ch2))
```

5. Écrire la fonction `patch(ch1,delta)` qui, à partir de la première chaîne et du *diff* calculé avec la fonction ci-dessus, re-calcule et renvoie la chaîne `ch2`.

```
>>> Δ = diff("Olivier", "Voilier")
>>> Δ
[(0, 'V'), (1, 'o'), (3, 'l')]
>>> patch("Olivier", Δ)
'Voilier'
>>> patch("Le rire sinistre", [(3,"p"), (8, "m")])
'Le pire ministre'
```

```
def patch(ch,d):
    tab = list(ch)
    for i in range(len(d)):
        (indice, lettre) = d[i]
        tab[indice] = lettre
    return tableau_vers_chaîne(tab)
```