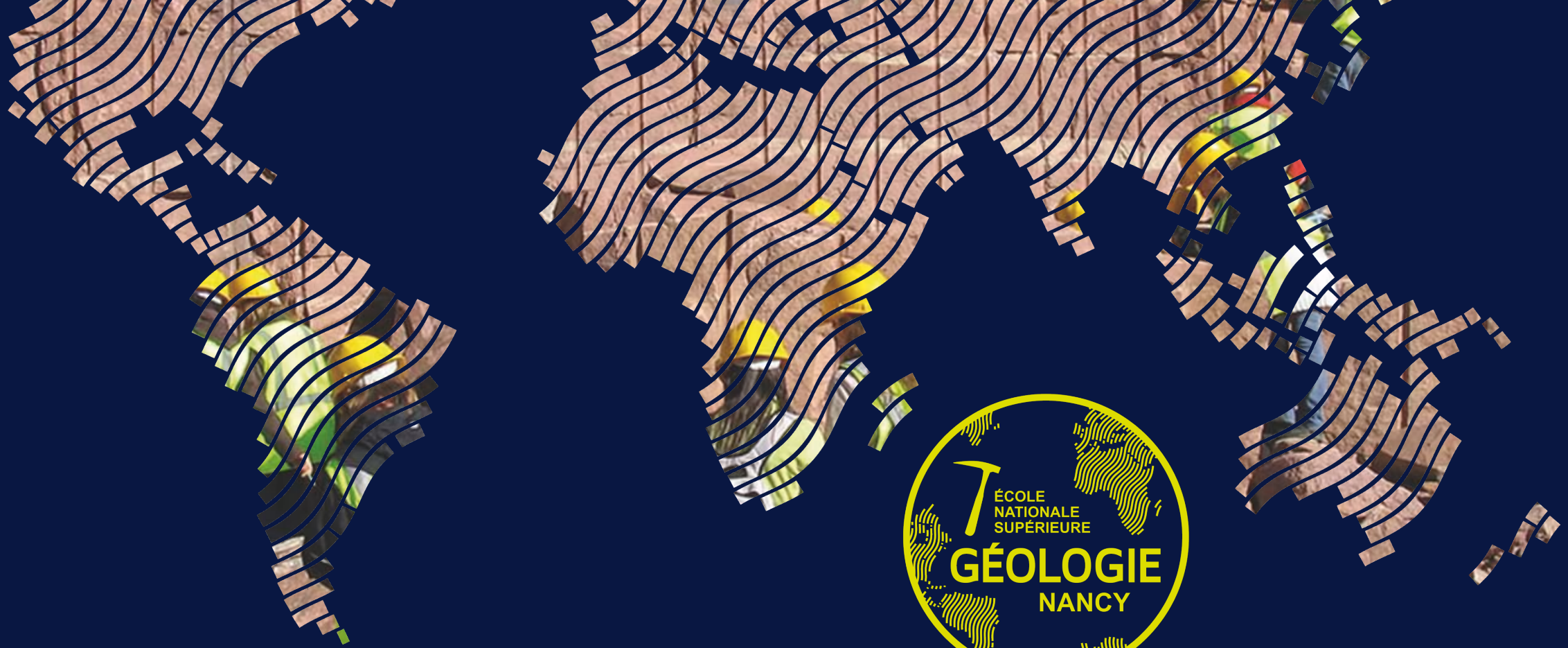


ÉCOLE NATIONALE SUPÉRIEURE GÉOLOGIE NANCY

RETOUR SUR EXAMEN INFO
SEMESTRE 5
2024-2025

Exercice I - Pythonesquerie

Exercice 1 - Pytonesqueries

```
7   def enigme(chaine, car, ind):
8       """
9
10      @param chaine:  chaîne de caractères
11      @param car:     caractère étudié
12      @param ind:     indice de départ dans la chaîne
13      @return:
14
15          Nombre d'occurrences consécutives
16          de car dans la chaîne à partir de la
17          position ind.
18
19      """
20      n = 0
21      for i in range(ind, len(chaine)):
22          if chaine[i] != car:
23              return n
24          n += 1
25
        return n
```

Cas à tester :

enigme("aabb", 'a', 0)

a	a	b	b

i = 0 ‘a’ = ‘a’ donc n = 1

i = 1 ‘a’ = ‘a’ donc n = 2

i = 2 ‘b’ != ‘a’ donc renvoie 2 FIN

enigme("aabcb", ‘b', 2)

a	a	b	c	b

i = 2 ‘b’ = ‘b’ donc n = 1

i = 3 ‘c’ != ‘b’ donc renvoie 1 FIN

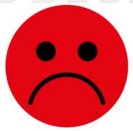
enigme(“caabb", 'a', 0)

c	a	a	b	b

i = 0 ‘c’ = ‘a’ donc renvoie 0 FIN

Exercice 1 - Pytonesqueries

```
7   def enigme(chaine, car, ind):
8       """
9
10      @param chaine: chaîne de caractères
11      @param car:    caractère étudié
12      @param ind:    indice de départ dans la chaîne
13      @return:
14
15          Nombre d'occurrences consécutives
16          de car dans la chaîne à partir de la
17          position ind.
18
19      """
20      n = 0
21      for i in range(ind, len(chaine)):
22          if chaine[i] != car:
23              return n
24          n += 1
25
        return n
```



- *Renvoie le rang / l'indice du premier caractère différent de car*
- *Renvoie le nombre d'occurrences de car à partir de la position ind*
- *Renvoie le nombre de caractères différents de car à partir de la position ind*
- *Renvoie le nombre d'occurrences de car dans la chaîne*

DOCSTRING ≠ COMMENTAIRES

Exercice 1 - Pytonesqueries

```
26
27  def mystere(ch, car1, car2):
28
29      n1 = enigme(ch, car1, 0)
30      n2 = enigme(ch, car2, n1)
31
32      if (n1 + n2) != len[ch]:
33          return False
34
35      return True
36
```

'builtin_function_or_method' object is not subscriptable

Une fonction n'est pas indexable. Indexation = [ ]

len[ch] => **len(ch)**

L'erreur est surlignée !

Exercice 1 - Pytonesqueries

```
36
37  def fonction_principale(lchaines, car1, car2):
38      i = 0
39      while i<len(lchaines):
40          if mystere(lchaines[i], car1, car2):
41              i+=1
42          else:
43              lchaines.remove(lchaines[i])
```

La fonction modifie la liste et ne renvoie rien.

ltests = ["aabbb", "abbb", "aabb", "bba", "a", "bb", "aabba", "aybb"]

1. `print(fonction_principale(ltests, 'a', 'b'))` — Affiche None : **Non cohérent**
2. `fonction_principale(ltests, 'a', 'b')` `print(ltests)` — Affiche la liste mise à jour : **Bonne réponse**
3. `c1 = 'a'` `c2 = 'b'` `fonction_principale (ltests, c1, c2)` `print(tab)` — tab non défini: **Erreur**
4. `lchaines = fonction_principale(ltests, 'a', 'b')` `print(lchaines)` — Affiche None : **Non cohérent**

UNIVERSITÉ DE LORRAINE
LORRAINE INP

Exercice II – Calcul d'une suite

Calcul d'une suite

- Soit la suite u suivante qui converge vers 1 : $u_i = \sqrt{\frac{1+ {u_{i-1}}^2}{2}}$

  avec u_0 compris entre -1 et 1

1) Ecrire une fonction ***suiteU*** qui, pour un u_0 et un *epsilon* donnés en paramètres, **retourne le premier *n* et la valeur de *u* correspondante** qui permette d'atteindre une valeur de la suite *u* proche de 1 à *epsilon* près

Définition de la fonction

```
def suiteU(u0, eps):  1 usage
    """..."""
    u = u0
    n = 0
    while (1-u) >= eps:
        n += 1
        u = sqrt((1 + u**2) / 2)

    return n, u
```

```
while (1-u0) >= eps:
    n += 1
    u = sqrt((1 + u0**2) / 2)
    u0 = u
```

Inutile d'utiliser une variable u et une variable u0

```
return n
```

n Lire l'énoncé

```
while (1-u) < eps:
```

Ne pas confondre condition d'arrêt et condition de continuation …

```
for i in range(...):
    while ...
```

Programme

2) Écrire le programme permettant de récupérer et d'afficher les résultats pour u0 = 0.5 et epsilon = 0.001

Programme = main → appel de fonction

```
if __name__ == '__main__':
    u0 = -2
    eps = 0.001
    n, u = suiteU(u0, eps)
    print("la suite obtient la précision demandée à : ", eps,
          "près, au rang : ", n)
```

Exercice III – Minéraux

Composant d'un minéral

1) Ecrire une fonction *compose*

- qui reçoit en données :
  - un symbole d'élément chimique (*ex : "Ca"*)
  - un minéral sous forme d'une liste de composants sous forme de couples ("*symbole-répétition*")
- et retourne *True* si l'élément chimique apparaît la liste des composants du minéral

Exemple : *compose("Ca", ["Ca-1", "S-1", "O-4", "H-2", "O-1"])* *retourne True*

compose("C", ["Ca-1", "S-1", "O-4", "H-2", "O-1"]) *retourne False*

- *Indication : on peut utiliser la fonction split :*

Exemples : *compose("Ca", ["Ca-1", "S-1", "O-4", "H-2", "O-1"]) retourne True*
compose("C", ["Ca-1", "S-1", "O-4", "H-2", "O-1"]) retourne False

Recherche d'un composant

```
def compose(symb, mineral):
    """..."""

    for comp in mineral:
        # on split chaque composant du minéral
        elt, rep = comp.split("-")
        if elt == symb:
            # on a trouvé le symbole cherché
            return True
    # si on n'a pas trouvé le symbole cherché
    # on retourné False
    return False
```

```
def compose(symb, mineral):
    """..."""
    lmineral = mineral.split('-')
    for comp in lmineral:
        if symb in lmineral:
            # on a trouvé le symbole cherché
            return True
    # si on n'a pas trouvé le symbole cherché
    # on retourné False
    return False
```

```
        if elt == symb:
            # on a trouvé le symbole cherché
            return True
        else:
            return False
```

S'arrête dès la 1ère condition non vérifiée
Or on ne peut décider de répondre False que lorsqu'on a examiné tous les éléments de la liste

Dictionnaire des minéraux

1) Soit un dictionnaire *mineraux* tel que : mineraux[nom_mineral] = composition chimique

- *Exemple :*
- *mineraux = {"**Gypse**" : ["Ca-1", "S-1", "O-4", "H-2", "O-1"], "**Orthose**" : ["K-1", "Al-1", "Si-3", "O-8"],*
- *"**Calcite**" : ["Ca-1", "C-1", "O-3"], "**Diamant**" : ["C-1"], "**Beryl**" : ["Be-3", "Al-2", "Si-6", "O-18"]}*

- En utilisant la fonction précédente, écrire une fonction *inventaire* qui retourne la liste des minéraux dont la composition chimique intègre ce symbole
- *Exemple : pour dico_mineraux et le symbole "Ca"*
  la fonction inventaire retourne la liste ["Gypse", "Calcite"]

Inventaire

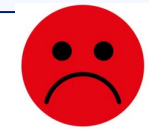
```
def inventaire(mineraux, symb):
    """..."""
    lmineraux = []
    for mineral in mineraux:
        if compose(symb, mineraux[mineral]):
            lmineraux.append(mineral)

    return mineral
```

```
def inventaire(mineraux, symb):
    """..."""
    lmineraux = []
    for mineral in mineraux:
        if compose(symb, mineral):
            lmineraux.append(mineral)

    return mineral
```

```
for mineral in mineraux:
    if compose(symb, mineraux[mineral]):
        lmineraux.append(mineraux[mineral])
```



Ajout d'un minéral

Ecrire une fonction *ajoute_mineral* :

- qui reçoit en entrée : un dictionnaire des minéraux, un nom de minéral, exemple "*Sylvanite*« , une liste de composants, exemple *[("Ag", "1"), ("Au", "1"), ("Te", "4")]*
- et qui ajoute ce minéral dans le dictionnaire : *exemple "Sylvanite" : ["Ag-1", "Au-1", "Te-4"]*

```
def ajoute_mineral(mineraux, mineral, composition):
    """..."""
    # mise dans le bon format de la composition du m
    composition_formatee = []
    for symb, rep in composition:
        composition_formatee.append(symb+rep)

    mineraux[mineral] = composition_formatee

    return mineraux
```

Ne pas oublier de mettre la composition dans le bon format

```
# on teste si le minéral apparaît
# déjà dans le dictionnaire
if mineral not in composition:
```

```
if mineral not in composition.keys():
```

```
return mineraux
```

Passage de paramètres par référence

Exercice IV – Carré magique

Carré magique

- Si n est un entier impair plus grand que 1, un carré magique d'ordre n est un carré contenant tous les entiers de 1 à n^2 disposés de façon à obtenir la même somme sur n'importe quelle ligne et n'importe quelle colonne.

  Par exemple, si n=5, on a le carré magique suivant :

0	17	24	1	8	15	=65
1	23	5	7	14	16	=65
2	4	6	13	20	22	=65
3	10	12	19	21	3	=65
4	11	18	25	2	9	=65
	=65	=65	=65	=65	=65	

On suppose qu'un carré est représenté par une liste de listes.

Exercice 4 – Carré magique

Écrire une fonction *somColonne* qui calcule la somme des éléments d'une colonne de numéro *numcol* dans un carre de taille *n* x *n*.

```
def somColonne(carre, numcol):

    somme = 0

    for ligne in carre:
        somme += ligne[numcol]

    return somme
```

```
def somColonne(carre, numcol):

    somme = 0

    for i in range(len(carre)):
        somme += carre[i][numcol]

    return somme
```

Inversion ligne et colonne

Exercice 4 – Carré magique

On suppose qu'on dispose d'une fonction somLigne similaire à la fonction précédente qui fait la somme des éléments d'une ligne. En utilisant ces deux fonctions, écrire une fonction estUnCarreMagique qui vérifie si un carre est magique ou non.

```
def estUnCarreMagique(carre):

    somme = somLigne(carre, 0)

    for i in range(len(carre)):

        if somLigne(carre, i) != somme:
            return False

        if somColonne(carre, i) != somme:
            return False

    return True
```

somme = (n**2 + 1) * n // 2

Vérification que toutes les lignes sont égales entre elles et que toutes les colonnes sont égales entre elles

Utilisation de or

Utilisation de listes de stockage

Comparaison ligne i à colonne i

Comparaison des moyennes, de la somme totale

Mauvais placement du return

Utilisation d'un compteur et return en fin

Deux boucles successives (lignes puis colonnes)

Double boucle (comparaison de toutes les lignes i et colonnes j (beaucoup de comparaison inutiles)