TD1 : AP

```
ens = {'a', 'b', 'c'}
```

1. Quel est le type de la valeur de `ens` ?
2. Comment obtenir le nombre d'éléments de `ens` ?
3. Comment tester si un élément est dans `ens` ?
4. Comment construire une liste contenant tous les éléments de `ens` ? Que dire de l'ordre des éléments de cette liste ?
5. Comment définir une variable dont la valeur est l'ensemble des voyelles (en version minuscules) ? (donnez deux réponses)
6. Comment obtenir l'intersection de `ens` avec l'ensemble des voyelles ? la réunion ?
7. On suppose définie la variable `texte` de la façon suivante :

```
texte = """La cigale et la fourmi
La cigale ayant chanté tout l'été """
```

8. Comment construire l'ensemble des caractères de la chaîne ? Quel est le nombre de caractères distincts de cette chaîne ?
9. Comment construire l'ensemble des voyelles contenues dans `texte` ?

1. ens est de type set
2. Il faut faire len(ens).
3. On fait le test el in ens.
4. Il faut faire list(ens). On ne peut pas savoir l'ordre de cette liste à l'avance
5. set("aeiouy") ou set(["a","e","i","o","u","y"])
6. ens.intersect(voyelles) ; ens.union(voyelles)
8. set(texte).len(set(texte)) == 20.
9. set(text).intersect(voyelles).

2 Dictionnaires

Si `d` est une variable dont la valeur est un dictionnaire, comment

10. consulter la valeur associée à une clé `k` ?
11. ajouter une association `k/v` ?
12. modifier une association ?
13. détruire une association de clé `k`?
14. obtenir la liste de toutes les clés ?
15. obtenir la liste de toutes les valeurs associées aux clés ?
16. obtenir la liste de toutes les associations contenues dans sous la forme de couples (clé, valeur) ?
17. construire un dictionnaire contenant les mêmes clés que `d` mais dont les valeurs associées sont toutes égales à `None`?

10. d[k]
11. d[k] = v
12. d[k] = nvle_valeur
13. del d[k].

14. `[k for k in d]`

15. `[v for v in d.values()]`

16. `[k,v for k,v in d.items()]`

17. `{k: None for k in d}`

3 Agenda téléphonique

Une liste nommée `CONNAISSANCES` contient des couples de chaînes de caractères (nom, numéro de téléphone). En voici un exemple

```
for nom, num in CONNAISSANCES:
        print(nom, num)


Calbuth 01-23-45-67-89
Talon 98-76-54-32-10
Cru 31-41-59-26-53
Lagaffe 35-62-95-14-13
```

18. Comment rechercher le numéro de téléphone d'une personne dont on connaît le nom dans cette liste ?

18.
```
def recherche_numéro(nom_recherche: str) -> str:
    for nom,num in CONNAISSANCES:
        if nom_recherche == nom:
            return num
    return ""
```

19. Construisez un dictionnaire contenant les associations (nom, numéro de téléphone) de la liste en supposant d'abord que la liste ne contient pas deux personnes de même nom.

20. Construisez un dictionnaire contenant les associations (nom, numéro de téléphone) de la liste en ne faisant plus cette hypothèse dans un deuxième temps. Proposez des idées pour résoudre ce problème.

19. Comme CONNAISSANCES est une liste de tuples, on peut faire `dict(CONNAISSANCES)`

20. On peut créer une fonction intermédiaire
```
def sans_doublons(liste: list[tuple[Any, Any]]) -> list[tuple[Any, Any]]:
    noms = set()
    res = []
    for nom, tel in liste
        if nom not in noms:
            noms.add(nom)
            res.append(nom, tel)
    return res
```

On peut donc simplement appliquer la fonction, ce qui nous rapporte au cas du 19.

4 Définitions en compréhension

4.1 Exemples simples

21. Donnez deux expressions en compréhension différentes dont la valeur est la liste multiples de trois positifs ou nuls et inférieurs ou égaux à 2024.
22. Soit `chaine` une chaîne de caractères. Donnez une expression en compréhension dont la valeur est la liste des préfixes de `chaine`.
23. Donnez une expression en compréhension dont la valeur est l'ensemble des facteurs de `chaine` (c'est à dire les sous-chaînes de la forme `chaine[debut:fin]` avec $0 \leq debut \leq fin \leq len(chaine)$).

On appelle *triplet pythagoricien* un triplet de nombres (a, b, c) tels que $a^2 + b^2 = c^2$.

24. Donnez une expression en compréhension dont la valeur est l'ensemble des triplets pythagoriciens d'entiers naturels non nuls tous inférieurs strictement à 100.

22. `[ chaine[0:i] for i in range(1, len(chaine)+1) ]`

23. `[chaine[i:j] for i in range(1, len(chaine+1)) for j in range(i, len(chaine+1))]`

24.
```
[
    a, b, c for a in range(100) for b in range(100) for c in range(100)
    if a*a + b*b == c*c
]
```

4.2 Dictionnaire d'occurences

Soit `chaine` une chaîne de caractères. On souhaite construire un dictionnaire d'occurences des caractères de cette chaîne.

25. Soit `carac` un caractère quelconque. Donnez une expression en compréhension dont la valeur est le nombre d'occurences de `carac` dans `chaine`.
26. Déduisez-en une expression en compréhension dont la valeur est le dictionnaire d'occurences des caractères de `chaine`.

25. `occurences = lambda char, chaine : len([c for c in chaine if c == char])`

26. `{c : occurences(c, chaine) for c in chaine}`

5 Doublons dans une liste

Appelons *doublon* dans une liste tout élément y figurant au moins deux fois. Ainsi la liste `[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]` contient trois doublons : `1` et `3` qui y figurent deux fois, et `5` qui y figure trois fois.

27. Réalisez un prédicat d'entête
`def a_des_doublons(liste: list[int]) -> bool`
qui renvoie `True` si cette liste possède au moins un doublon, et `False` dans le cas contraire. Vous en donnerez deux versions :

- une version n'utilisant aucune structure de données auxiliaire,
- une version utilisant un dictionnaire ou un ensemble.

La comparaison de ces recherches sera abordée dans un cours ultérieur.

28. Réalisez une fonction d'entête
`def liste_doublons(li: list[int]) -> list[int]:`
qui renvoie une liste des doublons d'une liste passée en paramètre.

27.

```
def a_des_doublons(liste : list[int]) -> bool :
    for i, el in enumerate(liste):
        if el in liste[:i] + liste[i+1:] :
            return True
    return False.
```

```
def a_des_doublons(liste : list[int]) -> bool :
    lettres = set()
    for el in liste :
        if el in lettres :
            return True
        else :
            lettres.add(el)
    return False
```

28.

```
def liste_doublons(li : list[int]) -> list[int]:
    lettres = set()
    doublons = []
    for el in liste
        if el in lettres:
            doublons.append(el)
        else :
            lettres.add(el)
    return doublons
```