Coanim 3- Nombres complexes

On se propose de découvrir ici une nouvelle structure de données python : les **dictionnaires**. Ils permettent de stocker des données dans une structure différente des listes pythons. On se sert ensuite de cette nouvelle structure pour créer des fonctions qui permettront de réaliser des calculs algébriques sur des nombres complexes.

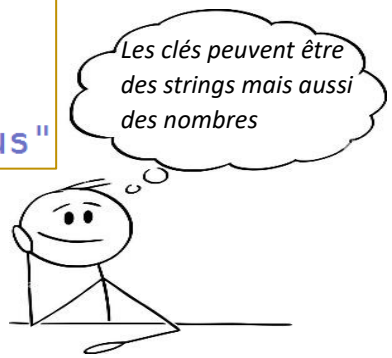
1- Point cours : Les dictionnaires en langage python

a. Creer un dictionnaire vide et le remplir :

Création d'un dictionnaire vide dans la variable **hanks**.

On remplit ce dictionnaire avec 4 valeurs identifiées avec les clés 'nom', 'prenom', 'naissance' et 'email'

```
hanks = {}
hanks["nom"] = "HANKS"
hanks["prenom"] = "Tom"
hanks["naissance"] = 1956
hanks["email"] = "tom.hanks@hollywood.us"
```



b. Creer un dictionnaire avec son contenu :

On peut aussi créer un dictionnaire de la façon suivante :

Création du dictionnaire contenant déjà 4 valeurs, dans la variable **wright**.

```
wright = {
    "nom": "WRIGHT",
    "prenom":"Robin",
    "dateNaissance" : 1966,
    "email" : "robin.wright@hollywwood.us"
}
```

c. Mixte des 2 methodes precedentes : On peut aussi mixer les 2 méthodes précédentes :

```
sinise = {
    "nom": "SINISE",
    "prenom":"Gary",
    "dateNaissance" : 1955
}
sinise["email"] = "gary.sinise@hollywwood.us"
```

d. Comment lire un element du dictionnaire :

```
>>> hanks["nom"]
'HANKS'
```

```
>>> hanks["email"]
'tom.hanks@hollywood.us'
```

Pour lire une valeur contenue dans un dictionnaire, on utilise la syntaxe :

nomDictionnaire[clé]

e. COMMENT MODIFIER UN ELEMENT DU DICTIONNAIRE :

```
>>> hanks["email"] = "tom.hanks@gmail.com"

>>> hanks["email"]
'tom.hanks@gmail.com'
```

f. COMMENT PARCOURIR LES ELEMENTS D'UN DICTIONNAIRE :

```
for cles in hanks :
    print(cles)
```

donne

```
>>> (executing file "cours.py")
nom
prenom
naissance
email
```

```
for cles in hanks :
    print(hanks[cles])
```

donne

```
>>> (executing file "cours.py")
HANKS
Tom
1956
tom.hanks@hollywood.us
```

g. COMMENT SAVOIR SI UNE CLE EST DANS UN DICTIONNAIRE :

```
>>> "nom" in hanks
True
```

```
>>> "telephone" in hanks
False
```

h. COMMENT SUPPRIMER DES VALEURS

```
>>> del forestGump["jenny"]
```

```
>>> forestGump.clear()
```

```
>>> del forestGump
```

2- APPLICATION POUR S'APPROPRIER LES DICTIONNAIRES :

On donne ci-contre les codes de 3 fonctions, accompagnés des lignes qui les exécutent. 2 lignes sont malheureusement à modifier.

```
def anglais() -> dict:
    dic = {}
    dic['je'] = 'i'
    dic['aime'] = 'love'
    dic['toi'] = 'you'
    return dic

def allemand() -> dict:
    d = {'je':'ich' , 'aime' : 'liebe' , 'toi' : 'du'}
    return 'à modifier'

def italien() -> dict:
    d = {}
    listeFrancais = ['je','aime','toi']
    listeItalien = ['io','amore','voi']
    for i in range(3) :
        cle = listeFrancais[i]
        d[cle] = 'à modifier'
    return d

# Programme principal
uk = anglais()
de = allemand()
it = italien()
print(uk,de,it)
```

⇨ Copier-coller ce code dans un fichier python que vous appellerez *traduction.py* .

⇨ Modifier ce fichier afin que le résultat de son exécution soit le suivant :

```
{'je': 'i', 'aime': 'love', 'toi': 'you'} {'je': 'ich', 'aime': 'liebe', 'toi': 'du'} {'je': 'io', 'aime': 'amore', 'toi': 'voi'}
```

On souhaite créer une fonction nommée *traductionMot()*. Elle prend en argument un string et un dictionnaire. En exécutant le programme principal ci-contre, on veut obtenir dans la console :

```
en anglais :  love
en allemand :  liebe
en italien :  amore
```

```
# Programme principal
uk = anglais()
de = allemand()
it = italien()
m = traductionMot("aime",uk)
print('en anglais : ',m)
m = traductionMot("aime",de)
print('en allemand : ',m)
m = traductionMot("aime",it)
print('en italien : ',m)
```

⇨ Compléter le fichier *traduction.py* en créant la fonction *traductionMot()*.

⇨ Une fois terminé, uploader le fichier *traduction.py* sur https://mathsapp-front.vercel.app/

3- APPLICATION SUR LES NOMBRES COMPLEXES : INTRODUCTION

Les nombres complexes en mathématiques contiennent 2 valeurs numériques : celle correspondant à la partie réelle et celle à la partie imaginaire. Par exemple le nombre complexe $z = -4 + i$ a une partie réelle qui est égale à -4 et une partie imaginaire égale à 1 : $Re(z) = -4$ et $Im(z) = 1$.

Pour stocker ces 2 valeurs numériques dans une même entité python, on peut utiliser un dictionnaire. Cela donne alors pour le nombre complexe $z = -4 + i$:

La partie **réelle** de ce nombre complexe est stockée avec la clé '**re'**

```
z = {"re": -4 , "im": 1}
```

ou

```
z = {}
z["re"] = -4
z["im"] = 1
```

La partie **imaginaire** de ce nombre complexe est stockée avec la clé

Pour afficher le nombre complexe proprement, on peut créer la fonction *affiche()* suivante :

Cette fonction a comme paramètre un dictionnaire que l'on nomme ici z

Cette fonction retourne un string

```
def affiche(z : dict) -> str :
    a = z["re"]
    b = z["im"]
    return f"({a} + {b} i)"
```

Cette fonction retourne un string reprenant la forme normalisée du nombre complexe mis entre parenthèses

Pour additionner 2 nombres complexes, on peut créer la fonction *somme()* suivante :

Cette fonction a comme paramètres 2 dictionnaires que l'on nomme ici z et Z

Cette fonction retourne un dictionnaire

```
def somme(z : dict, Z : dict) -> dict:
    a = z["re"]
    b = z["im"]
    A = Z["re"]
    B = Z["im"]
    s = {}
    s["re"] = a + A
    s["im"] = b + B
    return s
```

Attention, ne pas confondre z et Z

On retourne un dictionnaire dont la clé ''re'' a comme valeur la somme des parties réelles et la clé ''im'', la somme des parties imaginaires.

Si l'on utilise ces 2 fonctions avec par exemple, le programme principal suivant :

```
# Main
z = {"re": -4 , "im": 1}
Z = {"re": 2 , "im": 2}
s = somme(z , Z)
print(f"La somme de {affiche(z)} et {affiche(Z)} donne le nombre complexe {affiche(s)} ")
```

On obtient :

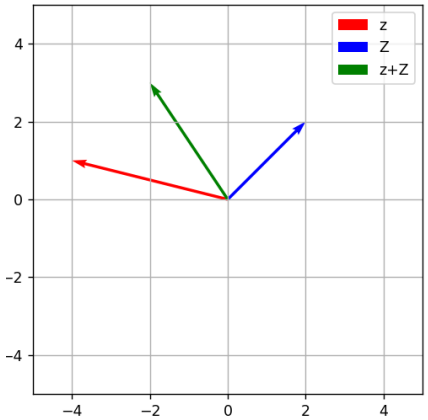
```
La somme de (-4 + 1 i) et (2 + 2 i) donne le nombre complexe (-2 + 3 i)
```

Pour visualiser le vecteur image d'un nombre complexe, on peut utiliser la librairie *matplotlib* (https://matplotlib.org/) Le code complet de la situation précédente est alors le suivant :

```
import matplotlib.pyplot as plt
def trace(z : dict , couleur : str , label : str) -> None :
    ax.quiver(0, 0, z["re"], z["im"], angles='xy', scale_units='xy', scale=1, color=couleur, label=label)

def affiche(z : dict) -> str :
    a = z["re"]
    b = z["im"]
    return f"({a} + {b} i)"

def somme(z : dict, Z : dict) -> dict:
    a = z["re"]
    b = z["im"]
    A = Z["re"]
    B = Z["im"]
    s = {}
    s["re"] = a + A
    s["im"] = b + B
    return s

# Main
fig, ax = plt.subplots() # ne pas modifier (pour matplotlib)
ax.set_aspect('equal') # ne pas modifier (pour matplotlib)

z = {"re": -4 , "im": 1}
trace(z,'red','z') # permet de tracer le vecteur image de z
Z = {"re": 2 , "im": 2}
trace(Z,'blue','Z') # permet de tracer le vecteur image de Z

s = somme(z , Z)
trace(s,'green','z+Z') # permet de tracer le vecteur image de z + Z
print(f"La somme de {affiche(z)} et {affiche(Z)} donne le nombre complexe {affiche(s)} ")

ax.set_xlim(-5, 5) # ne pas modifier (pour matplotlib)
ax.set_ylim(-5,5) # ne pas modifier (pour matplotlib)
ax.grid(True) # ne pas modifier (pour matplotlib)
ax.legend() # ne pas modifier (pour matplotlib)
plt.show() # ne pas modifier (pour matplotlib)
```

⇨ Copier-coller ce code dans un fichier que vous nommerez *complexe.py*.

⇨ Exécuter le code. Normalement on obtient la figure ci-contre, et dans la console, le texte ci-dessous :




```
La somme de (-4 + 1 i) et (2 + 2 i) donne le nombre complexe (-2 + 3 i)
```

4- APPLICATION SUR LES NOMBRES COMPLEXES : TRAVAIL A FAIRE

a. CREER UNE FONCTION QUI RETOURNE LE PRODUIT :

Rappel : Si $z = a + b\,i$ et $Z = A + B\,i$, alors :

$$z \times Z = (a + b\,i)(A + B\,i)$$

$$= a\,A + a\,B\,i + b\,A\,i + b\,B\,i^2$$

$$= a\,A + (a\,B + b\,A)i + b\,B \times (-1)$$

$$= (a\,A - b\,B) \quad + \quad (a\,B + b\,A)\,i$$

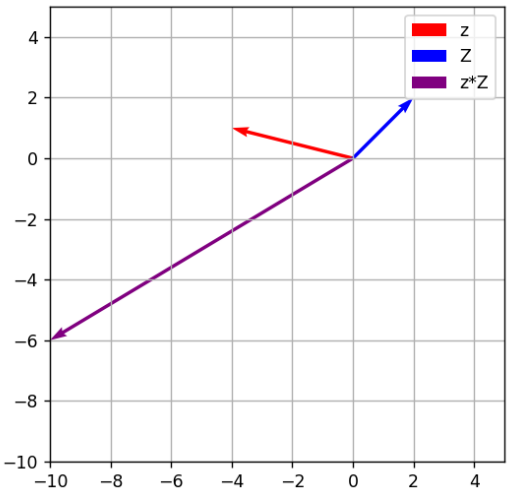
Partie réelle de $z\,z'$

Partie imaginaire de $z\,z'$

Exemple test : Si $z = -4 + i$ et $Z = 2 + 2i$,
Alors : $z \times Z = (-4 + i)(2 + 2i) = -8 - 8i + 2i + 2i^2 = -10 - 6i$
Avec le programme principal suivant :

```
# Main
fig, ax = plt.subplots()
ax.set_aspect('equal')

z = {"re": -4 , "im": 1}
trace(z,'red','z')
Z = {"re": 2 , "im": 2}
trace(Z,'blue','Z')

p = produit(z , Z)
trace(p,'purple','z*Z')
print(f"Le produit de {affiche(z)} et {affiche(Z)} donne le nombre complexe {affiche(p)} ")

ax.set_xlim(-10, 5)
ax.set_ylim(-10, 5)
ax.grid(True)
ax.legend()
plt.show()
```

On obtient la figure ci-contre :



et dans la console, le texte ci-dessous :

```
Le produit de (-4 + 1 i) et (2 + 2 i) donne le nombre complexe (-10 + -6 i)
```

⇨ Compléter le fichier *complexe.py* avec la fonction *produit()* qui retourne le nombre complexe égal au produit des 2 nombres complexes mis en argument.

b. CREER UNE FONCTION QUI RETOURNE LA CONJUGUE :

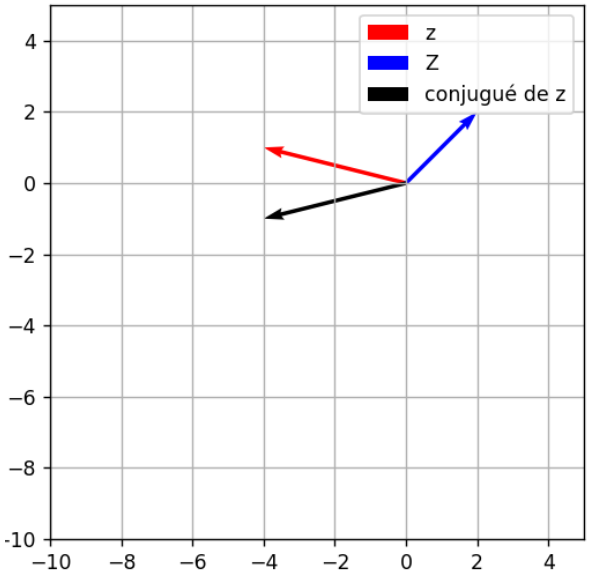
Rappel : Si $z = a + b\,i$, alors : $\bar{z} = a - b\,i$

Exemple test : Si $z = -4 + i$ alors : $\bar{z} = -4 - i$

Avec le programme principal suivant :

```
# Main
fig, ax = plt.subplots()
ax.set_aspect('equal')

z = {"re": -4 , "im": 1}
trace(z,'red','z')
Z = {"re": 2 , "im": 2}
trace(Z,'blue','Z')


c = conjugue(z)
trace(c,'black','conjugué de z')
print(f"Le conjugué de {affiche(z)} est le nombre complexe {affiche(c)} ")

ax.set_xlim(-10, 5)
ax.set_ylim(-10, 5)
ax.grid(True)
ax.legend()
plt.show()
```

On obtient la figure ci-contre :



Et dans la console, le texte ci-dessous :

```
Le conjugué de (-4 + 1 i) est le nombre complexe (-4 + -1 i)
```

⇨ Compléter le fichier *complexe.py* avec la fonction *conjugue()* qui retourne le nombre complexe égal au produit des 2 nombres complexes mis en argument.

c. CREER UNE FONCTION QUI RETOURNE LE QUOTIENT :

Rappel : Si $z = a + b\,i$ et $Z = A + B\,i$, alors :

$$\frac{z}{Z} = \frac{(a + b\,i)}{(A + B\,i)} = \frac{(a + b\,i)(A - B\,i)}{(A + B\,i)\,(A - B\,i)}$$

On a donc :

$$\frac{z}{Z} = \frac{z \times \bar{Z}}{Z \times \bar{Z}}$$

Exemple test : Si $z = -4 + i$ et $Z = 2 + 2i$,

Alors : $\frac{z}{Z} = \frac{(-4+i)}{(2+2i)} = \frac{(-4+i)(2-2i)}{(2+2i)(2-2i)} = \frac{-8+8i+2i-2i^2}{2^2+2^2} = \frac{-6+10i}{8}$

Partie réelle de $z\bar{Z}$ — Partie imaginaire de $z\bar{Z}$

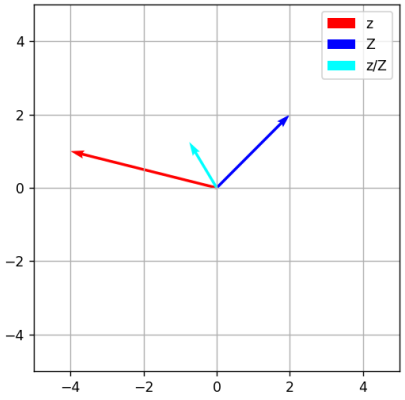
Alors : $\frac{z}{Z} = \frac{-6}{8} + \frac{10i}{8} = -0.75 + 1.25\ i$

Partie réelle de $Z\ \bar{Z}$, la partie imaginaire de $Z\ \bar{Z}$ est nulle

Avec le programme principal suivant :

```
# Main
fig, ax = plt.subplots()
ax.set_aspect('equal')

z = {"re": -4 , "im": 1}
trace(z,'red','z')
Z = {"re": 2 , "im": 2}
trace(Z,'blue','Z')

q = quotient(z , Z)
trace(q,'cyan','z/Z')
print(f"Le quotient de {affiche(z)} et {affiche(Z)} donne le nombre complexe {affiche(q)} ")

ax.set_xlim(-5, 5)
ax.set_ylim(-5, 5)
ax.grid(True)
ax.legend()
plt.show()
```

On obtient la figure ci-contre :



Et dans la console, le texte suivant :

```
Le quotient de (-4 + 1 i) et (2 + 2 i) donne le nombre complexe (-0.75 + 1.25 i)
```

⇨ Compléter le fichier *complexe.py* avec la fonction *quotient()* qui retourne le nombre complexe égal au quotient des 2 nombres complexes mis en argument.

5- UPLOAD DU FICHIER :

⇨ Une fois terminé, uploader le fichier *complexe.py* sur https://mathsapp-front.vercel.app/