Introduction

1. Premier pas avec Pyzo

Exercice 1. La fenêtre de `Pyzo` est essentiellement constituée de deux panneaux, un à droite pour l'édition de script et un en haut à gauche pour le shell permettant l'éxécution de commandes.

1. Sur le panneau correspondant au shell, cliquer sur `No shell selected` puis `Edit shell configuration`. Dans le menu déroulant `exe` choisir un interpréteur pour la version 3 de `Python`.
2. A l'adresse `http://www.lmpa.univ-littoral.fr/~fromentin/python`, télécharger le fichier `tp1.py` contenant le matériel nécessaire pour ce TP.
3. Enregistrer le fichier `tp1.py` sur votre compte et l'ouvrir au sein de `Pyzo`.

Les exercices de ce TP seront faits directement dans le fichier `tp1.py`, après les commentaires correspondants.

Exercice 2.

1. Exécuter la commande `Python` : `"Bonjour le monde !"`
2. Créer un script contenant la définition de fonction :

```
def dire_bonjour():
    print("Bonjour le monde !")
```

3. Enregistrer le script grâce à `Ctrl+S`.
4. Exécuter le script dans le shell grâce à `Ctrl+E`.
5. Exécuter la commande `dire_bonjour()`.

Exercice 3. Exécuter les suites de commandes suivantes et les commenter.

1. `>>> a=1  >>> a  >>> b=a  >>> b  >>> a=2  >>> a  >>> b`
2. `>>> a=[1,2]  >>> a  >>> b=a  >>> b  >>> a[0]=0  >>> a  >>> b`
3. `>>> a=[1,2]  >>> a  >>> b=a.copy()  >>> b  >>> a[0]=0  >>> a  >>> b`

Exercice 4. Ecrire une fonction `Entier res(Entier exp,E,Entier M)` permettant de calculer la somme $\sum_{i=1}^{M} i^{exp}$ pour $M \geqslant 1$ et $exp \geqslant 1$ donnés Calculer `res(3,10)`.

Exercice 5. La suite de Fibonacci F_n est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-2} + F_{n-1}$ pour $n \geqslant 2$.

1. Ecrire une fonction récursive `Entier fibo_rec(Entier n)` permettant de calculer F_n.
2. Ecrire une fonction non récursive `Entier fibo(Entier n)` permettant de calculer F_n.
3. Tester ces deux fonctions pour calculer F_{10}, F_{20} et F_{30}.

2. Nombres complexes

Exercice 6. Soit a un nombre positif. La suite u_n définie par $u_0 = 1$ et $u_n = 1/2(u_{n-1} + a/u_{n-1})$ pour $n \geqslant 1$ converge vers $\sqrt{a}$. Ecrire une fonction `racine_carree` permettant de donner une valeur approchée de $\sqrt{a}$ à 10^{-10} près où a est un nombre positif donné.

Le fichier `tp1.py` contient la définition d'une structure `Complexe` permettant de manipuler des nombres complexes. Nous pouvons créer un nombre complexe à l'aide des commandes `z=Complexe(2,3)` pour $z = 2 + 3\mathbf{i}$ ou `z=Complexe(-1)` pour $z = -1$. Si `z` est une variable de type `Complexe`, nous accédons à la partie réel de z grâce à `z.re` et à sa partie imaginaire grâce à `z.im`.

Exercice 7. Ecrire les fonctions suivantes :

1. `Complexe additionner(Complexe z1, Complexe z2)` additionnant deux nombres complexes.
2. `Complexe multiplier(Complexe z1, Complexe z2)` multipliant deux nombres complexes.
3. `Reel module(Complexe z)` retournant une valeur approchée du module d'un nombre complexe.

Exercice 8. Ecrire les fonctions suivantes :

1. `Complexe evaluer(Reel a, Reel b, Reel c, Complexe z)` retournant l'évaluation du polynôme $ax^2 + bx + c$ en le complexe z.
2. `Tableau<Complexe> racines(Reel a, Reel b, Reel c)` retournant l'ensemble des racines du polynôme $ax^2 + bx + c$.
3. Déterminer les racines de $x^2 - 1$, $x^2 + 2x + 1$, $x^2 + 1$, $x^2 - x - 1$ et $x^2 + x + 1$. Vérifier les résultats obtenus avec la fonction `evaluer`.

3. Tableaux et matrices

Pour connaitre la taille d'un tableau `tab` nous disposons de la commande `len(tab)`.

Exercice 9. Ecrire et tester (pour le tableau `tab` fourni) les fonctions suivantes :

1. `Entier min(Tableau<Entier> t)` retournant la plus petit élément du tableau `t`.
2. `Entier somme(Tableau<Entier> t)` la somme des éléments du tableau `t`.
3. `Entier nombre_pairs(Tableau<Entier> t)` le nombre d'éléments pair du tableau `t`.

Exercice 10. Le tri à bulle est un algorithme de tri (par ordre croissant ici) de tableau. Il consiste à comparer successivement les éléments consécutifs du tableau et à les permuter s'ils ne sont pas dans l'ordre. Après un premier passage le plus grand élément du tableau se retrouve forcément à la dernière place.

1. Trier à la main et à l'aide de l'algorithme de tri à bulle, le tableau `[2,5,3,1,3,6]`
2. Ecrire une fonction `tri_bulle(Tableau<Entier> t)` triant le tableau `t`
3. Tester votre fonction sur le tableau `tab2=[2,5,3,1,3,6]` puis sur le tableau `tab` de taille `20` fournis.

Une matrice à `l` lignes et `c` colonnes peut être représentée par un tableau de `l` tableaux de taille `c`. Les fonctions suivantes sont déjà fournies dans le fichier `tp1.py` :

- `affiche(Matrice<Entier> M)` affichant une matrice `A` ;
- `Entier nombre_lignes(Matrice<Entier> M)` retournant le nombre de lignes de `A`,
- `Entier nombre_colonnes(Matrice<Entier> M)` retournant le nombre de colonnes de `A`,
- `Matrice<Entier> creer_matrice(Entier l,Entier n)` retournant la matrice nulle à `l` lignes et `n` colonnes.

De même, des matrices `A`, `B` et `C` sont déjàs définies. Elles pourront être utilisées pour tester les différentes fonctions demandées.

Exercice 11. Ecrire les fonctions suivantes :

1. `Booleen est_nulle(Matrice<Entier> M)` testant si la matrice `M` est nulle.
2. `Matrice<Entier> additionner_matrice(Matrice<Entier> M,Matrice<Entier> N)` retournant la matrice $M + N$.
3. `Matrice<Entier> multiplier_matrice(Matrice<Entier> M,Matrice<Entier> N)` retournant la matrice $M \times N$.