Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique-Informatique	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2024/2025 Filière : SM, SP et ST Niveau d'étude : 2ème année Semestre : 1 Nombre de pages : 3 Date : 24/10/2024 Durée : 1H30

Devoir Surveillé n°1 d'informatique

Exercice1 : (10 points)

Dans cet exercice on cherche à calculer une **approximation** de la fonction **exponentielle de x** par l'expression suivante : $\mathbf{e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!}}$ **(Equ.1)**

D'où pour tout entier « n » positif on a : $\mathbf{S(x,n) = \sum_{i=0}^{n} \frac{x^i}{i!}}$ **(Equ.2)**

La somme « **S(x,n)** » est une approximation de la fonction mathématique exponentielle « $\mathbf{e^x}$ » dont la précision dépend de la valeur de « **n** » tel que : $\lim_{n\to\infty} S(x,n) = e^x$ **(Equ.3)**

NB : Pour la suite de l'exercice il est interdit d'utiliser la fonction « **`exp()`** » prédéfinie dans le module math ou d'autres modules.

.

(1 pts) 1- Ecrire une fonction python « **`Saisie`** » qui saisit et retourne à partir du clavier un entier positif « $n \geq 0$ » avec tous les contrôles et traitements d'exceptions nécessaires.

(1 pts) 2- Ecrire « **`factIter(n)`** » une fonction Python **itérative** qui calcule et retourne le factoriel de n « **n !** » dont voici un exemple d'appel :

```
>>> factIter(4)
24
```

(1 pts) 3- Déduire une fonction **récursive** « **`factRec(n)`** » qui calcule le factoriel de n « **n !** ». Dans ce cas ajouter un traitement d'exception pour contrôler « n » qui doit être positive « **`n≥0`** ».

```
>>> factRec(4)
24
>>> factRec(-4)
AssertionError: Erreur1: n=-4 < 0 !
  …
```

(1 pts) 4- Donner le schéma d'exécution de la fonction récursive « **`factRec(n)`** » pour **n = 5**. Déduire la complexité **spatiale** et la complexité **temporelle**.

(1 pts) 5- Ecrire une fonction **récursive** « **`puiRecRap(x,n)`** » qui exploite les deux identités suivantes « Equ.4 » pour calculer rapidement la puissance « $\mathbf{x^n}$ » :

$$\begin{cases} x^{2k} = (x^k)^2 \\ x^{2k+1} = x\,(x^k)^2 \end{cases} \quad \textbf{(Equ.4)}$$

(1 pts) 6- A partir de l'**equ.2** définir une fonction **récursive** « **`expRec(x,n)`** » qui calcule une approximation de l'exponentielle de **x**, avec **n** doit être un entier positif. Dans ce cas utiliser les fonctions de calcul de la puissance et celle de calcul de factoriel précédentes.

(1 pts) 7- Ecrire une fonction **itérative** « **`expEps(x,eps=0.001)`** » qui calcule une approximation de « $\mathbf{e^x}$ » avec comme condition d'arrêt d'itération de la boucle quand le terme $|\frac{x^i}{i!}| \leq eps$. Cette fonction retourne un tuple **`(ex,n)`** dont le premier élément est un réel qui représente l'approximation de « $\mathbf{e^x}$ » et le deuxième élément est un entier qui représente le nombre d'itérations « **`n = i`** » effectué. Voici un exemple d'appel de cette fonction :

```
>>> expEps(1)
(2.7180555555555554, 7)
>>> expEps(4)
(54.59706179769672, 15)
>>> expEps(4,0.01)
(54.59398264287153, 14)
>>> expEps(-2)
(0.1350970017636685, 10)
```

8- Donnez le code en python d'un programme principal qui permet :

(1 pts)

a) D'ouvrir un fichier texte en lecture « **in.txt** » qui contient des réels sur une ou plusieurs lignes comme l'indique l'exemple suivant :

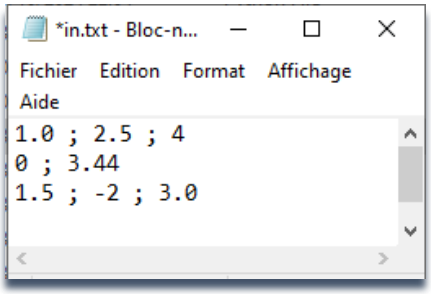

Fig.1 : Exemple de fichier « in.txt »

Et charger le contenu de ce fichier dans une liste « **Lx** » avec un réel par élément de la liste. Dans notre exemple **Lx = [1.0, 2.5, 4.0, 0.0, 3.44, 1.5, -2.0, 3.0]**

(1 pts)

b) Calculer les exponentielles des tous les réels de liste « **Lx** » et stoker les résultats dans un dictionnaire « **d** » dont les **clés** sont les réels de « **Lx** » et les **valeurs** sont les approximations des exponentielles calculées par la fonction « **expRec(x,n)** » avec « **n** » sera saisie par l'utilisateur à travers la fonction « **Saisie** ». Dans notre exemple de test et pour une saisie de n = 10 : **d = {1.0: 2.7182818011463845, 2.5: 12.18174319124306, 4.0: 54.44310405643739, 0.0: 1.0, . . .}**

(1 pts)

c) Sauver le contenu du dictionnaire « **d** » dans un fichier texte « **out.txt** » en deux colonnes « **x** » et « **exp(x)** » avec la représentation des valeurs réels **sur 7 chiffres dont 3 après la virgule** comme l'indique la figure suivante :

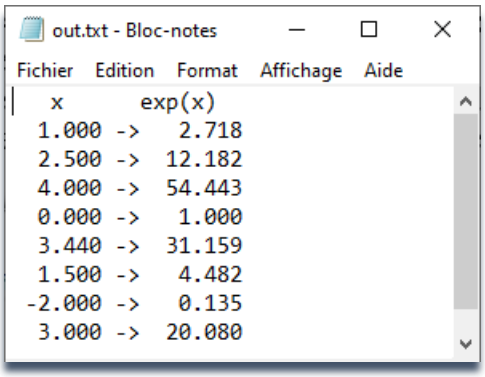

Fig.2 : Exemple de fichier « out.txt »

NB : voici un exemple d'un affichage d'un réel sur **6 chiffres dont 2 après la virgule** en utilisant le **f-string** :

```
>>> x = 2.7182818011463845
>>> print(f"{x=:6.2f}")
x=  2.72
>>>
```

Exercice2 : (10 points)

L'objectif de cet exercice est la modélisation des formes géométriques avec la POO sous Python.

(1 pts)

1. Créer une classe fondamentale Python nommée « **Point** », définie par deux attributs d'instance, **x** et **y**, représentant les coordonnées d'un point dans un plan.

(1 pts)

2. Définir une la méthode spéciale « **__str__** » qui permet la conversion d'un objet de type **Point** en une chaîne de caractères dont voici un exemple d'utilisation :

```
>>> p1 = Point(4,-3)
>>> print(p1)
P(4,-3)
```

(1 pts)

3. Définir une méthode « **`distance`** » qui retourne la distance euclidienne entre ce point et le centre de repère O(0,0).

```
>>> p1 = Point(4,-3) ; p1.distance()
5.0
```

(1 pts)

4. Définir la méthode « **`symétrie`** » qui retourne le point (objet de type **Point**) symétrique par rapport au centre de repère O :

```
>>> p1_sym = p1.symetrie()
>>> print(p1_sym)
P(-4,3)
```

(1 pts)

5. Créer une classe fondamentale nommée Rectangle, définie par les attributs suivants :
   - C : un attribut de type Point représentant le centre du rectangle,
   - L : la largeur du rectangle,
   - H : la hauteur du rectangle.

Voici un exemple d'instanciation d'un objet nommée « **boite** » de type **Rectangle** :

```
>>> p = Point(4,-3)  #sera utilisé comme centre du rectangle
>>> boite = Rectangle(p,4,2) #largeur = 4 et hauteur = 2
```

(1 pts)

6. Définir une méthode « **`perimSurf`** » qui retourne un dictionnaire avec le périmètre et la surface du rectangle :

```
>>> boite.perimSurf()
{'Perimetre': 12, 'Surface': 8}
```

(1 pts)

7. Définir la méthode « **`coins`** » qui retourne une liste de quatre objets de type **Point** relatifs aux quatre coins du rectangle à partir du coin supérieur gauche et avec un parcours dans le sens d'aiguilles d'une montre. Comme l'exemple suivant :

```
>>> L=boite.coins()
>>> for p in L: print(p) #p est un objet de type Point.
P(2.0,-2.0)
P(6.0,-2.0)
P(6.0,-4.0)
P(2.0,-4.0)
```

(1 pts)

8. Définir une méthode « **`affiche`** » qui affiche les caractéristiques d'un objet de type « **Rectangle** » comme l'indique l'exemple suivant :

```
>>> boite.affiche()
Rectangle :
     centre = P(4,-3)
     Largeur = 4
     Hauteur = 2
     Périmètre = 12.000
     Surface = 8.000
     Coins = P(2.0,-2.0),P(6.0,-2.0),P(6.0,-4.0),P(2.0,-4.0)
```

(2 pts)

9. Ecrire un programme principal qui fait le traitement suivant :
   - a- Saisie de deux réels « **x** et **y** » comme coordonnées d'un point « **c** »,
   - b- Saisie de la largeur « L » et la hauteur « H » d'un rectangle (deux réels strictement supérieurs à 0),
   - c- Création de l'objet « **c** » de type **Point**,
   - d- Création d'un objet « **rect** » de type **Rectangle** avec le centre « **c** », de largeur « **L »** et de hauteur « **H »**,
   - **e-** Affichage des caractéristiques de ce rectangle « **rect** » avec sa méthode « **affiche** ».

Bon courage…