Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Sfax
Institut Préparatoire aux Etudes
d'Ingénieurs de Sfax

IPEIS

وزارة التعليم العالي والبحث العلمي
جامعة صفاقس
المعهد التحضيري للدراسات الهندسية
بصفاقس

Année universitaire : 2019-2020

Devoir & Examen semestre 2

Matière : informatique

Classe : MP2, PC2 et PT2 — Nombre de pages : 4

Date : Juillet 2020 — Durée : 2h

Devoir

On relève dans certains jours de l'année les degrés de température. Une observation sera une liste de 3 valeurs [jour , mois , degré]. Par exemple, une température de 20°C enregistrée le 13 février correspond à la liste [13, 2, 20]. Pour simplifier on supposera que toutes les observations sont enregistrées pendant des années non bissextiles (le nombre de jours du mois de février d'une année non bissextile est égal à 28).

On suppose que les listes *Nb_jours* et *Mois* suivantes sont préalablement définies :

Nb_jours=[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

Mois=['Janvier','Février','Mars','Avril','Mai','Juin','Juillet','Août','Septembre','Octobre',
'Novembre','Décembre']

1. Ecrire une fonction *Mois_jour()* qui, à partir des deux listes Nb_jour et Mois, retourne un dictionnaire qui associe à chaque mois de la liste *Mois* le nombre de jours correspondant.

   Valeur retournée de la fonction Mois_jour():{"Janvier" :31,"Fevrier" :28, ...,"Décembre":31}

2. Ecrire une fonction *ordre_Mois()* retournant un dictionnaire qui associe à chaque mois de la liste *Mois*, son numéro dans l'année.

   Valeur retournée de la fonction ordre_Mois(): {"Janvier" :1,"Fevrier" :2, ...,"Décembre":12}

3. Ecrire une fonction *valide(obs)* qui retourne True si l'observation *obs* correspond à un jour et un mois valide et False sinon. (Mois valide : compris entre 1 et 12. Jours valide : compris entre 1 et le nombre de jours max du mois).

4. Ecrire la fonction *Num_jour* qui permet de retourner le numéro du jour dans l'année d'une observation *obs* donnée et supposée valide.

   Exemple : l'observation [12, 2, 10] enregistrée le 12 février correspond au 43^{ième} jour de l'année.

5. Ecrire la fonction *Avant* qui, à partir de deux observations *obs1* et *obs2* supposées valides, retourne True si l'observation *obs1* a eu lieu avant l'observation *obs2* et retourne False sinon. (utiliser impérativement la fonction *Num_jour*)

6. Ecrire la commande qui ordonne une liste d'observations *L* par ordre chronologique des observations. (utiliser la commande prédéfinie *sorted*)

N.B :

sorted(liste, key = f) *: tri suivant les valeurs que la fonction f prend sur les éléments de la liste. La fonction donnant la clé de tri peut être une fonction prédéfinie dans un module (par exemple la fonction sin du module math), une fonction définie précédemment par le programmeur (par def), ou une fonction définie sur place (par lambda).*

Exemple :

```
>>> s = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
>>> sorted(s, key=lambda x: x[2])
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Examen

Exercice 1 *(10 pts)*

1. Définir une classe nommée *Complexe* permettant d'instancier des objets représentant les nombres complexes et possédant les méthodes suivantes :

- Un constructeur permettant d'initialiser les attributs x et y d'instance. Ces attributs représentent respectivement la partie réelle et imaginaire d'un nombre complexe. (valeurs par défaut : 0),
- une méthode *module* retournant le module du nombre complexe,
- la surcharge de la méthode spéciale *sub* qui permet de soustraire deux nombres complexes,
- une méthode *distance* retournant la distance entre deux points. (Chacun des points est défini par un nombre complexe),
- la surcharge d'une méthode spéciale d'affichage dont l'invocation sur une instance d'attributs x= 1.2 et y=2.3, permet d'afficher : « Complexe (1.2, 2.3) ».

Dans la suite, un polygone sera défini par une instance de la classe *Polygone*. L'attribut points de cette instance est une liste d'objets de la classe *Complexe*. Cette liste représente les sommets du polygone.

2. Définir une classe *Polygone* avec les méthodes suivantes :

- Un constructeur permettant d'initialiser les sommets d'un polygone et un attribut nom='polygone',
- une méthode *est_ferme* permettant de retourner *True* si le contour du polygone est fermé, *False* sinon,
- une méthode *ferme_polygone* permettant d'ajouter un complexe qui ferme le polygone,
- une méthode *perimetre* permettant de fermer un polygone s'il n'est pas fermé et retourner le périmètre de ce polygone (somme des distances entre les points). Utiliser la méthode *distance* de la classe *Complexe*,
- une méthode d'affichage qui permet d'afficher le nom du polygone et son périmètre.

3. Définir une classe *Triangle* qui hérite de la classe *Polygone* avec un constructeur permettant d'initialiser trois attributs représentant les sommets du triangle et qui surcharge l'attribut d'instance nom='triangle'.

On rappelle qu'un polygone régulier à n côtés inscrit dans un cercle (C), de centre O et de rayon r, à des sommets dont les coordonnées sont :

$$x_k = r\cos(\frac{2k\pi}{n}) \quad \text{et} \quad y_k = r\sin(\frac{2k\pi}{n}) \text{ pour } k = 0 .. n$$

4. Ecrire une fonction *liste_points* qui retourne une liste contenant des objets de la classe *Complexe* définissant la liste des points d'un polynôme régulier à n côtés.

5. Afficher le périmètre d'un polygone régulier de cinq côtés et d'un triangle inscrits dans un cercle de centre O et de rayon 2.

Exercice 2 *(10 pts)*

On donne trois tables concernant les gouvernorats, les délégations et les municipalités de Tunisie (une municipalité appartient à une délégation et une délégation à un gouvernorat).

Gouvernorats (num_gouv, nom)

Delegations (num_del, #num_gouv, nom)

Municipalites (num_muni, #num_del, nom, nbr_habitants, surface)

N.B : tout attribut souligné désigne une clé primaire et tout attribut précédé par # désigne une clé étrangère.

Donner les requêtes SQL permettant de :

1. Donner le nombre total d'habitants de la Tunisie.
2. Donner le nombre de municipalités dans la délégation "Sfax_sud".

Requêtes avec jointures

3. Donner la liste des noms des délégations des gouvernorats "Mannouba" et "Ariana".
4. Donner sans doublons la liste des noms des délégations contenant une municipalité dont le nom commence par "Ben".

Requêtes imbriquées

5. Donner la liste des noms des municipalités dont la population excède 10 fois la population moyenne des différentes municipalités. (AVG calcule la moyenne)
6. Que donne chacune des requêtes suivantes :

a) **SELECT DISTINCT** Gouvernorats.nom
FROM Delegations **JOIN** Gouvernorats **ON** Delegations.num_gouv = Gouvernorats.num_gouv
WHERE Delegations.nom LIKE 'B%'

b)**SELECT** nom **FROM** Gouvernorats
WHERE EXISTS (***SELECT*** * ***FROM*** *Delegations* ***WHERE*** *Delegations.num_gouv = Gouvernorats.num_gouv* ***AND*** *Delegations.nom* ***LIKE*** *'B%'*)

N.B : La commande EXISTS s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête.