RÉPUBLIQUE TUNISIENNE --- MINISTÈRE DE L'ÉDUCATION	**EXAMEN DU BACCALAURÉAT** **SESSION 2026**	
	Épreuve Pratique : **Algorithmique et Programmation**	
	Section : **Sciences de l'informatique**	
	Coefficient de l'épreuve : **1**	Durée : **1h 30 mn**

Le sujet comporte 5 pages numérotées de 1/5 à 5/5

Important :

1) *Toutes les ressources à utiliser se trouvent dans le dossier "Ressources" situé sur la racine du disque C.*
2) *Il est demandé au candidat :*
   - *De créer, dans le dossier Bac2026 situé sur la racine du disque C, un dossier de travail portant son numéro d'inscription (6 chiffres) et dans lequel il doit enregistrer, au fur et à mesure, tous les fichiers solution de ce sujet.*
   - *De copier, dans son dossier de travail, le fichier "**Algo1.rar**" situé dans "C:\Ressources", puis d'extraire son contenu, en utilisant le mot de passe "**3uzs6c**", dans ce même dossier de travail.*
   - *De s'assurer que le contenu extrait est formé des fichiers "**AjoutObjet.py**", "**InterfAjout.ui**", "**InterfManipulation.ui**" et "**Manip.py**".*
   - *De vérifier à la fin de l'épreuve que tous les fichiers sont dans son dossier de travail.*

Un musée souhaite informatiser la gestion de son stock d'objets archéologiques. Dans les anciens registres les quantités des objets sont exprimées en chiffres romains.

On souhaite développer une application Python munie de deux interfaces graphiques qui permet :

- de saisir la désignation d'un objet archéologique et sa quantité en chiffres romains,
- de convertir la quantité en chiffres romains en son équivalent décimal,
- d'enregistrer l'objet dans un fichier d'enregistrements,
- de consulter et d'afficher de manière triée les objets enregistrés.

On supposera qu'un nombre romain est une chaine de caractères non vide, composée de chiffres romains choisis uniquement parmi les lettres suivantes : **I**, **V**, **X**, **L**, **C**, **D** et **M** et qu'elle ne contient pas plus de trois lettres identiques consécutives.
Chaque chiffre correspond à une valeur décimale comme indiqué dans le tableau ci-dessous :

Chiffre romain	I	V	X	L	C	D	M
Valeur correspondante	1	5	10	50	100	500	1000

Pour convertir un nombre romain en un nombre décimal, on parcourt la chaîne de gauche à droite en appliquant les règles suivantes :

- Si un chiffre romain a une valeur **supérieure ou égale** à celle du chiffre suivant, on **ajoute** sa valeur.
- Si un chiffre romain a une valeur **strictement inférieure** à celle du chiffre suivant, on **retranche** sa valeur.
- Si un chiffre romain est en dernière position, on **ajoute** sa valeur.

Exemples :

- **CXCII** vaut **192** en décimal. En effet on a : **C** > **X** on ajoute 100, **X** < **C** on retranche 10, **C** > **I** on ajoute 100, **I** = **I** on ajoute 1 et enfin on ajoute 1 qui correspond à la valeur de la dernière lettre du nombre romain. Donc on aura : 100 – 10 + 100 + 1 + 1 = **192**
- **XLVI** vaut **46** en décimal. En effet on a : **X** < **L** on retranche 10, **L** > **V** on ajoute 50, **V** > **I** on ajoute 5 et enfin on ajoute 1 qui correspond à la valeur de la dernière lettre du nombre romain. Donc on aura : –10 + 50 + 5 + 1 = **46**

- **MDCCL** vaut **1750** en décimal. En effet on a : **M** > **D** on ajoute 1000, **D**>**C** on ajoute 500, **C=C** on ajoute 100, **C > L** on ajoute 100 et enfin on ajoute 50 qui correspond à la valeur de la dernière lettre du nombre romain. Donc on aura :1000 + 500+100 + 100 + 50 = **1750.**

Travail demandé :

1- En se basant sur la figure **Fig.1** de l'**annexe**, compléter l'interface graphique "**InterfAjout.ui**" par les éléments manquants (un objet **LineEdit** et un **bouton**).

2- Apporter les modifications nécessaires au fichier "**AjoutObjet.py**" afin de réaliser les actions suivantes :

a. Développer le module **RomainToDecimal** qui s'exécute suite au clic sur le bouton **"Convertir"** et qui permet :

- **d'afficher** un message d'erreur via une boite de dialogue **QMessageBox** si la quantité saisie n'est pas un nombre romain valide (voir figure **Fig.2**).
  NB : Faire appel à la fonction **Valider** déjà développée pour vérifier la validité de la quantité saisie.
- **de convertir** cette quantité en décimal et de l'**afficher** dans le label nommé **QD** si elle est valide (voir figure **Fig.3**).

b. Développer le module **Ajouter** qui s'exécute suite au clic sur le bouton **"Ajouter"** et qui permet :

- **d'afficher** un message d'erreur via une boite de dialogue **QMessageBox** si aucun nom d'objet archéologique n'a été saisi (voir la figure **Fig.4**),
- **d'afficher** un message d'erreur via une boite de dialogue **QMessageBox** si le label nommé **QD** est vide (voir la figure **Fig.5**),
- **de récupérer**, dans le cas contraire, le nom de l'objet archéologique, la quantité en chiffres romains et la quantité en décimal et **d'ajouter** ces informations dans le fichier "**Objets.dat**". Chaque enregistrement de ce fichier est composé des champs suivants :
  - **Objet** : représente le nom de l'objet archéologique.
  - **QR** : représente la quantité en chiffres romains.
  - **QD** : représente la quantité en décimal équivalente à **QR**.

NB :

- Pour récupérer le contenu d'un label, utiliser la méthode **text()**.
- Le fichier "**Objets.dat**" doit être créé dans le dossier de travail.
- Après l'ajout d'un nouvel enregistrement, le module doit vider les deux objets **LineEdit** ainsi que le label **QD**, puis afficher un message, via une boite de dialogue **QMessageBox**, indiquant que l'ajout a été effectué avec succès (voir la figure **Fig.6**).

c. Compléter la partie exploitation de l'interface graphique "**InterfAjout.ui**" par les noms adéquats afin :

- **de charger** l'interface graphique depuis le fichier "**InterfAjout.ui**",
- **d'appeler** le module nommé **RomainToDecimal** suite au clic sur le bouton intitulé "**Convertir**",
- **d'appeler** le module nommé **Ajouter** suite au clic sur le bouton intitulé "**Ajouter**".

3- En se basant sur la figure **Fig.7** de l'**annexe**, compléter l'interface graphique "**InterfManipulation.ui**" par les éléments manquants (un **label** et un objet **ListWidget**).

4- Apporter les modifications nécessaires au fichier "**Manip.py**" afin de réaliser les actions suivantes :

a. Développer le module **Recherche** qui s'exécute suite au clic sur le bouton **"Rechercher"** et qui permet :

- **de récupérer** le nom de l'objet archéologique,
- **d'afficher**, si l'objet existe dans le fichier "**Objets.dat**", ses quantités en chiffres romains et décimaux dans les labels correspondants nommés **QR** et **QD** (voir figure **Fig.8**),
- **d'afficher**, si l'objet n'existe pas dans le fichier "**Objets.dat**", un message d'erreur via une boite de dialogue **QMessageBox** (voir figure **Fig.9**).

b. Implémenter le module **Tri** ci-dessous, qui permet de trier récursivement les éléments d'un tableau d'enregistrements **T** de la case d'indice **i** à la case d'indice **j-1** dans l'ordre croissant du champ **QD**.

```
Procédure Tri (@T : Tab, i, j : Entier)
DEBUT
    Si T[i].QD >T[j-1].QD Alors
            V←T[i]
            T[i]←T[j-1]
            T[j-1]←V
    FinSi
    Si  (j - i) > 2 Alors
            k← (j-i) div 3
            Tri (T, i, j-k)
            Tri (T, i+k, j)
            Tri (T, i, j-k)
    FinSi
FIN
```

NB : Chaque enregistrement du tableau **T** a la même structure qu'un enregistrement du fichier "**Objets.dat**".

c. Développer le module **TriAffiche** qui s'exécute suite au clic sur le bouton "**Trier et afficher**" et qui permet :

- **d'appeler** le module **Tri** afin de trier le contenu du fichier "**Objets.dat**" dans l'ordre croissant du champ **QD**, puis **de sauvegarder** le contenu trié dans le même fichier,
- **d'afficher** dans l'objet "**ListWidget**" le contenu du fichier "**Objets.dat**", trié dans l'ordre croissant du champ **QD**.
  Selon l'option sélectionnée dans la liste déroulante, chaque ligne de l'objet "**ListWidget**" doit contenir :
  - soit le nom de l'objet suivi de sa quantité en chiffres romains, puis de sa quantité en décimale (voir la figure **Fig.10**) ;
  - soit le nom de l'objet suivi de sa quantité en décimale (voir la figure **Fig.11**).

  NB : Utiliser les caractères "**- - -**" pour séparer les différentes valeurs de chaque ligne de l'objet "**ListWidget**", conformément aux figures **Fig.10** et **Fig.11**.
- **d'afficher** un message d'erreur conformément à la figure **Fig.12** lorsque l'option "**Choisir une valeur**" est sélectionnée dans la liste déroulante.

d. Compléter la partie exploitation de l'interface graphique "**InterfManipulation.ui**" par les noms adéquats afin :

- **de charger** l'interface graphique depuis le fichier "**InterfManipulation.ui**",
- **d'appeler** le module nommé **Recherche** suite au clic sur le bouton intitulé "**Rechercher**",
- **d'appeler** le module nommé **TriAffiche** suite au clic sur le bouton intitulé "**Trier et afficher**".

Annexe :

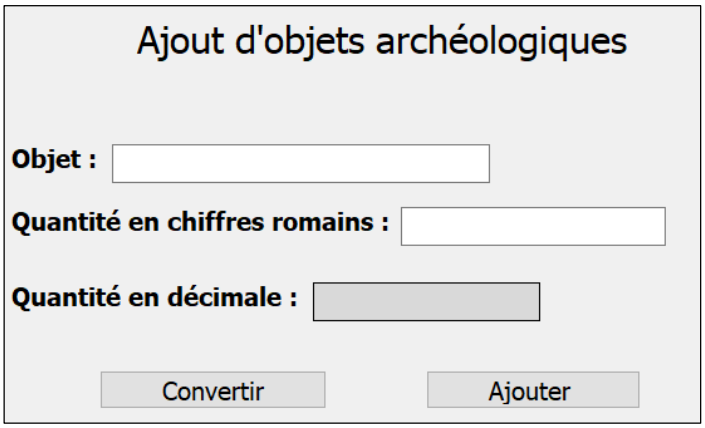

Fig.1

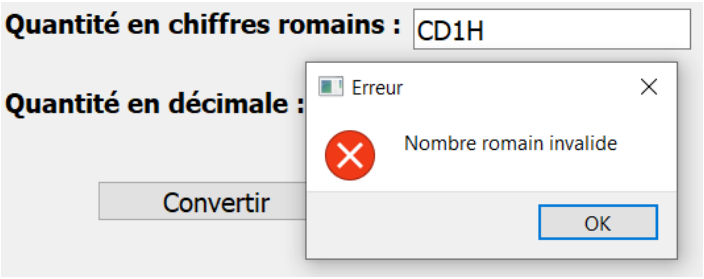

Fig.2

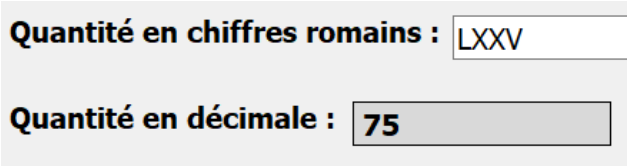

Fig.3

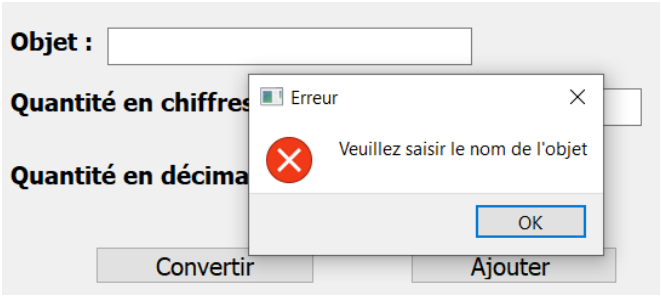

Fig.4

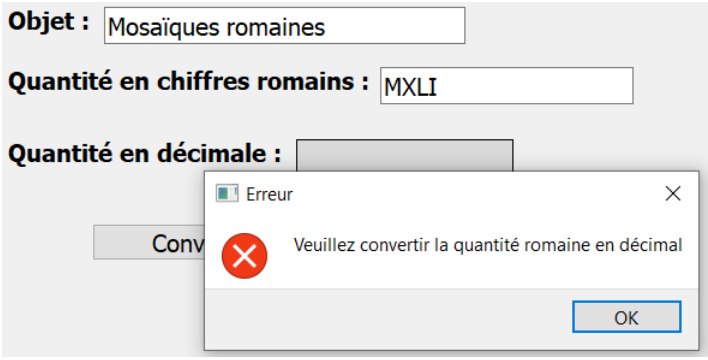

Fig.5

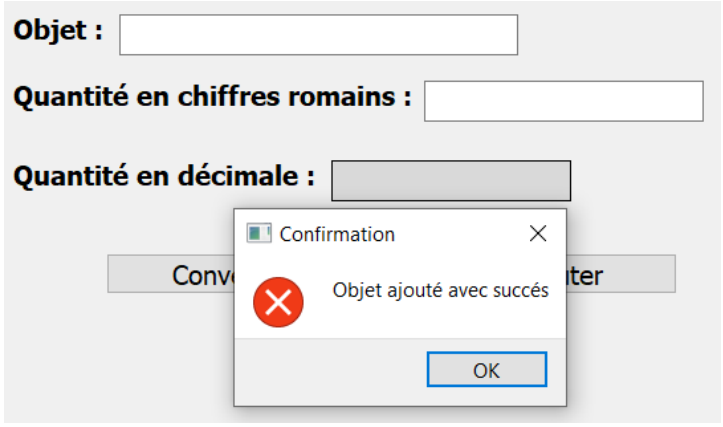

Fig.6

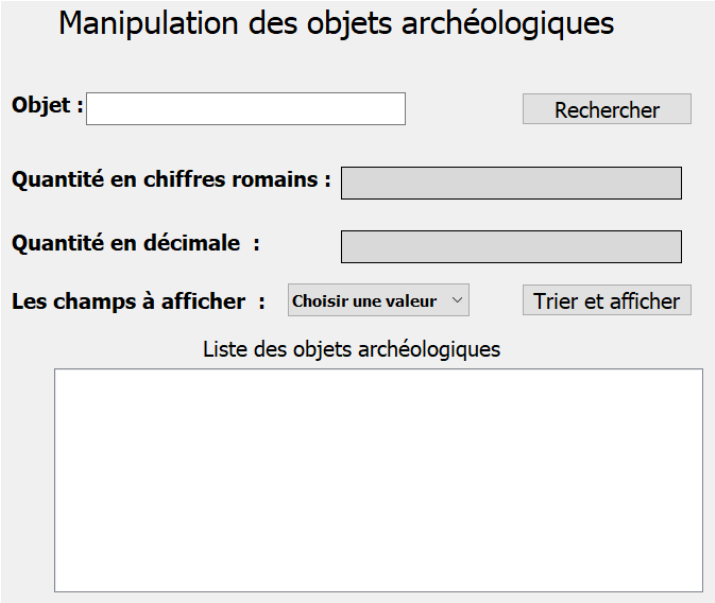

Fig.7

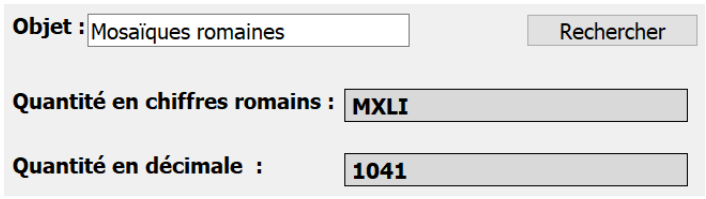

Fig.8

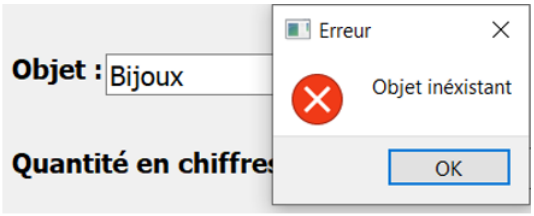

Fig.9

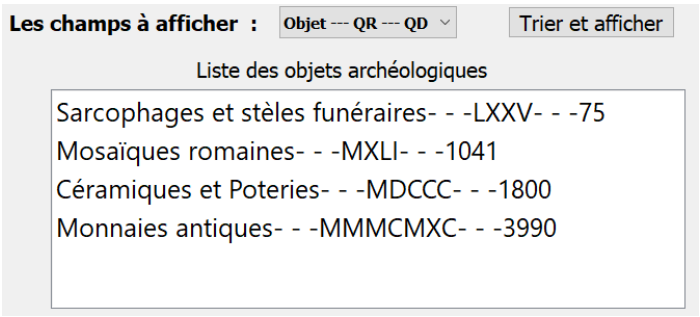

Fig.10

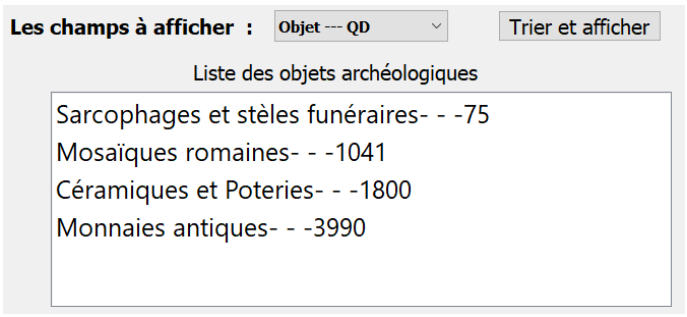

Fig.11

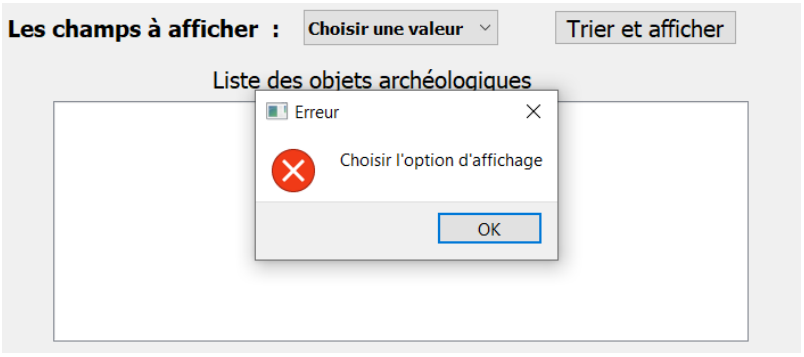

Fig.12

Grille d'évaluation :

	Traitement	Nombre de points
1	Complétion de l'interface graphique "**InterfAjout.ui**"	1
2	Modification dans "**AjoutObjet.py**"	8
3	Complétion de l'interface graphique "**InterfManipulation.ui**"	1
4	Modification dans "**Manip.py**"	10