université de BORDEAUX	**UE : 4TPM101U** / Initiation à l'informatique **Épreuve :** Test de contrôle continu n°5 **Date :** Lundi 05 Décembre 2022 **Durée :** 10 minutes Documents : Aucun document	Collège Sciences et Technologies

Nom : **Prénom :**

N.B. : Répondez directement sur la feuille

Exercice 1 : Analyse de code

Une erreur s'est glissée dans ce programme. Entourez la et corrigez la.

```
def nbElementsAvant(s:str, c:str) -> int :
  cpt = 0
  i = 0
  while s[i] != c :
    cpt +=1
  return cpt
```

Que fait cette fonction ?

Exercice 2 : Écriture de code

Écrivez une fonction `fonction(seuil: int) -> int` qui retourne le plus petit entier n tel que $n + n^2 + n^3 + n^4 + ... + n^n$ dépasse `seuil`.

Exercice 3 : Graphes

À l'aide de la documentation fournie, écrivez la fonction `colorierAutresSommets(G: graphe, nom: str, c1: color, c2: color, c3: color)` qui :

- colore le sommet désigné par `nom` dans la couleur `c1` passée en argument ;
- colore tous les voisins du sommet désigné par `nom` dans la couleur `c2` passée en argument ;
- colore tous les autres sommets du graphe `G` dans la couleur `c3` passée en argument.

C.4 Utilisation de la bibliothèque de graphes

Il faut commencer par importer le module `bibgraphes` :

```
from bibgraphes import *
```

Attention à respecter la distinction entre minuscules et majuscules :

L'argument `G` est un graphe	
`listeSommets(G:graphe) -> list`	retourne la *liste* des *sommets* de `G`
`nbSommets(G:graphe) -> int`	retourne le *nombre* de *sommets* de `G`
`sommetNom(G:graphe,etiquette:str) -> sommet`	retourne le *sommet* de `G` désigné par son *nom* (`etiquette`)
`afficherGraphe(G:graphe)` ou simplement `dessiner(G)`	demande (très poliment) au logiciel *Graphviz* de dessiner le graphe `G` ; voir page suivante pour les détails

L'argument `s` est un sommet	
`listeVoisins(s:sommet) -> list`	retourne la *liste* des *voisins* de `s`
`degre(s:sommet) -> int`	retourne le *degré* de `s`
`nomSommet(s:sommet) -> str`	retourne le *nom* (étiquette) de `s`
`colorierSommet(s:sommet,c:str)`	colorie `s` avec la couleur `c`. Exemples de couleurs : `'red'`, `'green'`, `'blue'`, `'white'`, `'cyan'`, `'yellow'`
`couleurSommet(s:sommet) -> str`	retourne la *couleur* de `s`
`marquerSommet(s:sommet)` `demarquerSommet(s:sommet)`	marque le sommet `s` démarque le sommet `s`
`estMarqueSommet(s:sommet) -> bool`	retourne `True` si `s` est marqué, `False` sinon
`listeAretesIncidentes(s:sommet) -> list`	retourne la *liste* des arêtes *incidentes* à `s`

L'argument `a` est une arête	
`nomArete(a:arete) -> str`	retourne le *nom* (étiquette) de `a`
`marquerArete(a:arete)` `demarquerArete(a:arete)`	marque l'arête `a` démarque l'arête `a`
`estMarqueeArete(a:arete) -> bool`	retourne `True` si `a` est marquée, `False` sinon

Arguments : un sommet `s` et une arête `a`	
`sommetVoisin(s:sommet,a:arete) -> sommet`	retourne le sommet voisin de `s` en suivant l'arête `a`

L'argument `u` est une liste	
`melange(u:list) -> list`	retourne une copie mélangée aléatoirement de la liste `u`. Exemple : `melange(listeSommets(G))` où `G` contient un graphe
`elementAleatoireListe(u:list)`	retourne un élément choisi aléatoirement dans la liste `u` si celle-ci est non vide. Si la liste `u` est vide la fonction retourne une erreur `IndexError`. Exemple : `elementAleatoireListe(listeSommets(G))` où `G` contient un graphe

Les fonctions suivantes permettent de construire des graphes de taille variable :

`construireComplet(n:int) -> graphe`	retourne le graphe complet K_n Exemple : `K5 = construireComplet(5)`
`construireBipartiComplet(m:int,n:int) -> graphe`	retourne $K_{m,n}$. Exemple : `K34 = construireBipartiComplet(3,4)`
`construireArbre(d:int,h:int) -> graphe`	retourne l'arbre de hauteur h dont chaque sommet possède d fils. Exemple : `arbre = construireArbre(2,3)`
`construireGrille(m:int,n:int) -> graphe`	retourne la grille rectangulaire avec m lignes et n colonnes. Exemple : `grille = construireGrille(4,6)`
`construireTriangle(n:int) -> graphe`	retourne la grille triangulaire d'ordre n Exemple : `t5 = construireTriangle(5)`

La fonction `ouvrirGraphe` permet d'ouvrir un graphe existant au format .dot. Le format .dot est très standard et utilisé très largement pour sauvegarder des graphes, en fait c'est un simple fichier texte, vous pouvez l'ouvrir pour voir à quoi il ressemble ! On peut également écrire des graphes, notamment pour sauvegarder un coloriage.

`ouvrirGraphe(nom:str) -> graphe`	Ouvre le fichier *nom* et retourne le graphe contenu dedans. (par exemple `ouvrirGraphe("fichier.dot")`.
`ecrireGraphe(G:graphe, nom:str)`	Sauvegarde le graphe `G` dans le fichier *nom* (par exemple `ecrireGraphe(G:graphe, "fichier.dot")`.

La fonction `afficherGraphe` (aussi appelée `dessiner`) comporte des paramètres facultatifs :

`afficherGraphe(G:graphe,True)`	dessine le graphe `G` en affichant les noms (étiquettes) des arêtes
`afficherGraphe(G:graphe,algo='neato')`	dessine le graphe `G` en utilisant un algorithme où les arêtes sont traitées comme des ressorts
`afficherGraphe(G:graphe,algo='circo')`	dessine le graphe `G` en utilisant un algorithme de placement des sommets sur un cercle

université de BORDEAUX	**UE : 4TPM101U** / Initiation à l'informatique **Épreuve :** Test de contrôle continu n°5 **Date :** Lundi 05 Décembre 2022 **Durée :** 10 minutes Documents : Aucun document	Collège Sciences et Technologies

Nom : **Prénom :**

N.B. : Répondez directement sur la feuille

Exercice 1 : Analyse de code

Une erreur s'est glissée dans ce programme. Entourez la et corrigez la.

```
def nbElementsApres(s:str, c:str) -> int :
  cpt = 0
  i = len(s)-1
  while s[i] != c :
    cpt +=1
  return cpt
```

Que fait cette fonction ?

Exercice 2 : Écriture de code

Écrivez une fonction `fonction(seuil: int) -> int` qui retourne le plus petit entier n tel que $1^2 + 2^2 + 3^2 + ... + n^2$ dépasse `seuil`.

Exercice 3 : Graphes

À l'aide de la documentation fournie, écrivez la fonction `colorierAutresSommets(G: graphe, nom: str, c1: color, c2: color, c3: color)` qui :

- colore le sommet désigné par `nom` dans la couleur `c3` passée en argument ;
- colore tous les voisins du sommet désigné par `nom` dans la couleur `c2` passée en argument ;
- colore tous les autres sommets du graphe `G` dans la couleur `c1` passée en argument.

C.4 Utilisation de la bibliothèque de graphes

Il faut commencer par importer le module `bibgraphes` :

```
from bibgraphes import *
```

Attention à respecter la distinction entre minuscules et majuscules :

L'argument `G` est un graphe	
`listeSommets(G:graphe) -> list`	retourne la *liste* des *sommets* de `G`
`nbSommets(G:graphe) -> int`	retourne le *nombre* de *sommets* de `G`
`sommetNom(G:graphe,etiquette:str) -> sommet`	retourne le *sommet* de `G` désigné par son *nom* (`etiquette`)
`afficherGraphe(G:graphe)` ou simplement `dessiner(G)`	demande (très poliment) au logiciel *Graphviz* de dessiner le graphe `G` ; voir page suivante pour les détails

L'argument `s` est un sommet	
`listeVoisins(s:sommet) -> list`	retourne la *liste* des *voisins* de `s`
`degre(s:sommet) -> int`	retourne le *degré* de `s`
`nomSommet(s:sommet) -> str`	retourne le *nom* (étiquette) de `s`
`colorierSommet(s:sommet,c:str)`	colorie `s` avec la couleur `c`. Exemples de couleurs : `'red'`, `'green'`, `'blue'`, `'white'`, `'cyan'`, `'yellow'`
`couleurSommet(s:sommet) -> str`	retourne la *couleur* de `s`
`marquerSommet(s:sommet)` `demarquerSommet(s:sommet)`	marque le sommet `s` démarque le sommet `s`
`estMarqueSommet(s:sommet) -> bool`	retourne `True` si `s` est marqué, `False` sinon
`listeAretesIncidentes(s:sommet) -> list`	retourne la *liste* des arêtes *incidentes* à `s`

L'argument `a` est une arête	
`nomArete(a:arete) -> str`	retourne le *nom* (étiquette) de `a`
`marquerArete(a:arete)` `demarquerArete(a:arete)`	marque l'arête `a` démarque l'arête `a`
`estMarqueeArete(a:arete) -> bool`	retourne `True` si `a` est marquée, `False` sinon

Arguments : un sommet `s` et une arête `a`	
`sommetVoisin(s:sommet,a:arete) -> sommet`	retourne le sommet voisin de `s` en suivant l'arête `a`

L'argument `u` est une liste	
`melange(u:list) -> list`	retourne une copie mélangée aléatoirement de la liste `u`. Exemple : `melange(listeSommets(G))` où `G` contient un graphe
`elementAleatoireListe(u:list)`	retourne un élément choisi aléatoirement dans la liste `u` si celle-ci est non vide. Si la liste `u` est vide la fonction retourne une erreur `IndexError`. Exemple : `elementAleatoireListe(listeSommets(G))` où `G` contient un graphe

Les fonctions suivantes permettent de construire des graphes de taille variable :

`construireComplet(n:int) -> graphe`	retourne le graphe complet K_n Exemple : `K5 = construireComplet(5)`
`construireBipartiComplet(m:int,n:int) -> graphe`	retourne $K_{m,n}$. Exemple : `K34 = construireBipartiComplet(3,4)`
`construireArbre(d:int,h:int) -> graphe`	retourne l'arbre de hauteur h dont chaque sommet possède d fils. Exemple : `arbre = construireArbre(2,3)`
`construireGrille(m:int,n:int) -> graphe`	retourne la grille rectangulaire avec m lignes et n colonnes. Exemple : `grille = construireGrille(4,6)`
`construireTriangle(n:int) -> graphe`	retourne la grille triangulaire d'ordre n Exemple : `t5 = construireTriangle(5)`

La fonction `ouvrirGraphe` permet d'ouvrir un graphe existant au format .dot. Le format .dot est très standard et utilisé très largement pour sauvegarder des graphes, en fait c'est un simple fichier texte, vous pouvez l'ouvrir pour voir à quoi il ressemble ! On peut également écrire des graphes, notamment pour sauvegarder un coloriage.

`ouvrirGraphe(nom:str) -> graphe`	Ouvre le fichier *nom* et retourne le graphe contenu dedans. (par exemple `ouvrirGraphe("fichier.dot")`.
`ecrireGraphe(G:graphe, nom:str)`	Sauvegarde le graphe `G` dans le fichier *nom* (par exemple `ecrireGraphe(G:graphe, "fichier.dot")`.

La fonction `afficherGraphe` (aussi appelée `dessiner`) comporte des paramètres facultatifs :

`afficherGraphe(G:graphe,True)`	dessine le graphe `G` en affichant les noms (étiquettes) des arêtes
`afficherGraphe(G:graphe,algo='neato')`	dessine le graphe `G` en utilisant un algorithme où les arêtes sont traitées comme des ressorts
`afficherGraphe(G:graphe,algo='circo')`	dessine le graphe `G` en utilisant un algorithme de placement des sommets sur un cercle