Numéro d'anonymat :

Université de Bordeaux
Département Licence, **J1MI1003**

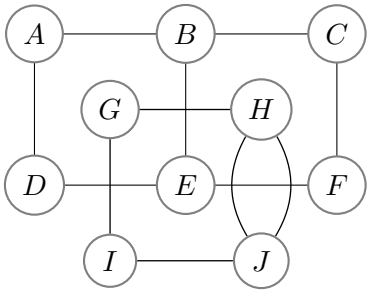
Examen du 11 janvier 2016
11h-12h30

Le sujet comporte 5 exercices et 7 pages, dont une page d'annexe

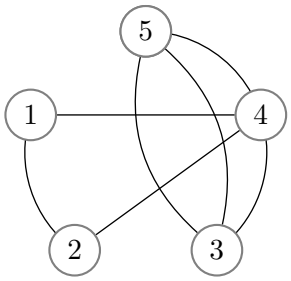
Aucun document n'est autorisé – Toutes les fonctions de manipulation d'images et de graphes disponibles sont rappelées en annexe page 7. Vous pouvez détacher cette annexe pour plus de facilité.

Exercice 1 Les questions de cet exercice portent toutes sur ces trois graphes : G_0, G_1 et G_2.

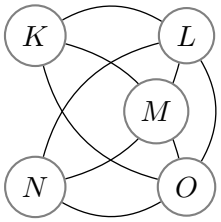
Graphe G_0



Graphe G_1



Graphe G_2



1. Remplir le tableau ci-dessous en n'oubliant pas de justifier votre réponse lorsque c'est nécessaire.

	G_0	G_1	G_2
Est-il connexe ? (si non, justifier)			
Est-il simple ? (si non, justifier)			
Donner un sommet de degré maximal et préciser son degré.			
Donner la longueur maximale des cycles élémentaires.			

Exercice 2 *Fonctions `python` sur les graphes. Dans cet exercice les graphes sont simples.*

1. Écrire une fonction `estDansLaListe(x, L)` qui renvoie `True` si l'élément `x` appartient à la liste `L`, et qui renvoie `False` sinon.

2. Écrire une fonction `estVoisinDAuMoinsUn(x, L)` qui renvoie `True` si le sommet `x` n'est pas dans la liste `L` et est voisin d'au moins un des sommets de `L`, et qui renvoie `False` sinon. Pour écrire cette fonction, vous pourrez tirer avantage à utiliser la fonction précédente.

On définit le **voisinage commun** de deux ensembles de sommets U et V d'un même graphe G, comme étant l'ensemble des sommets qui ne sont ni dans U ni dans V mais qui sont à la fois voisins d'au moins un des sommets de U et d'au moins un des sommets de V.

3. Écrire une fonction `nbVoisinageCommun(G, U, V)` qui renvoie le nombre de sommets qui sont dans le voisinage commun des listes de sommets `U` et `V`. Vous pourrez tirer avantage à utiliser la fonction précédente.

Exercice 3 L'objectif de cet exercice est d'écrire le code de deux fonctions Python permettant d'effectuer la transformation miroir (symétrie axiale) d'une image. Voici un exemple :

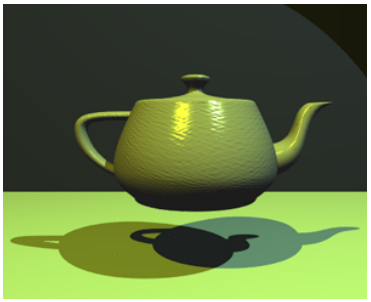
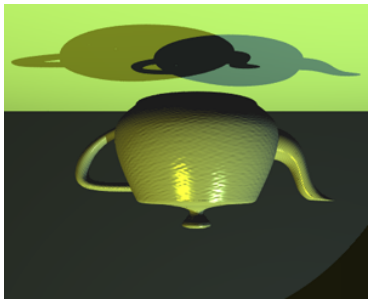
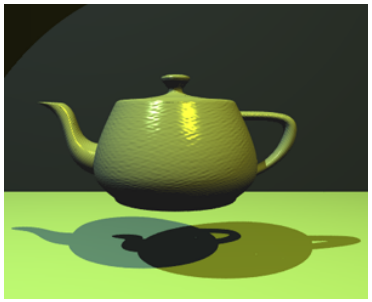


image initiale


miroir vertical


miroir horizontal

1. Écrire la fonction `miroirVertical(img1, img2)` qui place dans `img2` l'image correspondant au miroir vertical de `img1` (`img1` et `img2` sont supposées de même taille).

2. Écrire la fonction `miroirHorizontal(img)` qui crée et retourne une nouvelle image correspondant au miroir horizontal de `img`.

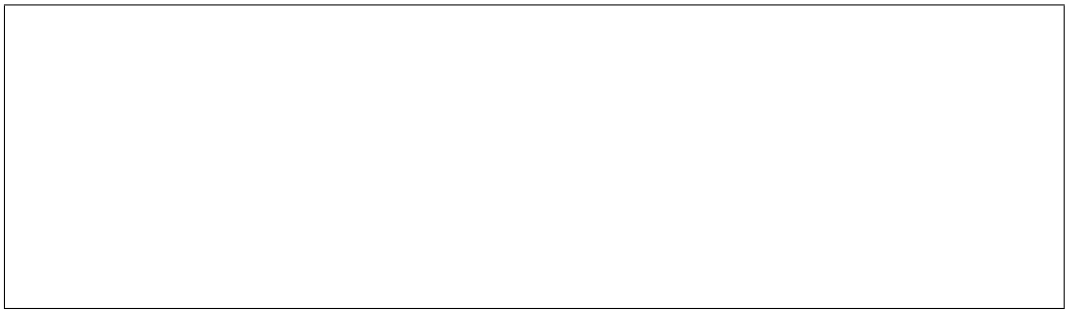
Exercice 4 1. Dessiner un arbre avec 2 sommets de degré 3, 3 sommets de degré 2 et des feuilles.

2. Combien l'arbre que vous venez de dessiner a-t-il de feuilles ?

3. Établir une formule reliant le nombre n de sommets d'un arbre à la somme s des degrés de ses sommets.

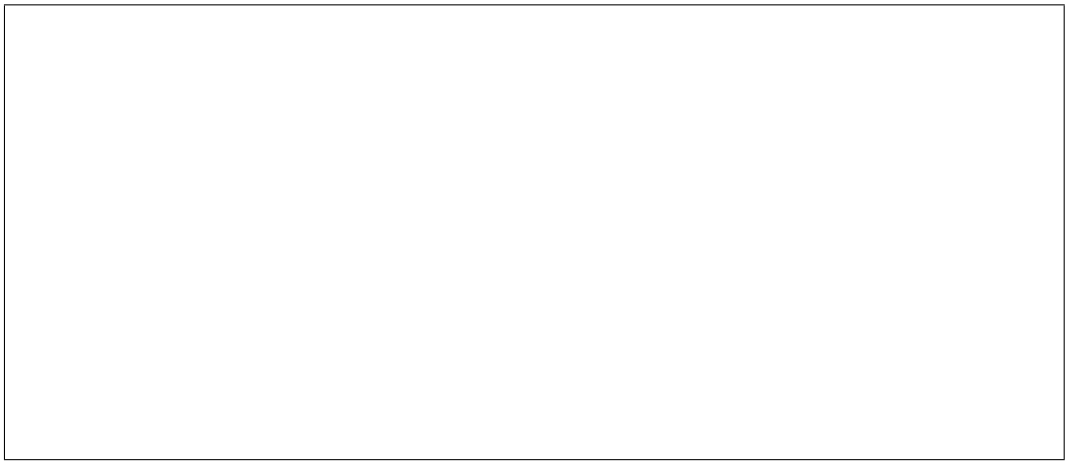
4. Un arbre possède a sommets de degré 3, b sommets de degré 2 et tous ses autres sommets sont des feuilles. Exprimer le nombre de feuilles de cet arbre en fonction de a et b.

Exercice 5 1. Écrire une fonction `facteurImpair(n)`, qui, étant donné un entier naturel `n` non-nul, renvoie le plus grand diviseur impair de `n`. Le résultat sera obtenu par une succession de divisions de `n` par 2 tant que `n` est pair. Par exemple, `facteurImpair(504)` retournera 63, puisque 504 est pair, 252 (504 divisé par 2) et 126 (252//2) sont pairs, et 63 (126//2) est impair.



2. Écrire une fonction `puissanceDiviseur(p,n)`, qui, étant donné un entier naturel `n` non-nul, renvoie la plus grande puissance de `p` qui divise `n`.

   Par exemple, `puissanceDiviseur(2,504)` retournera `8`, puisque 504 est divisible par $8 = 2^3$, mais pas par $16 = 2^4$.



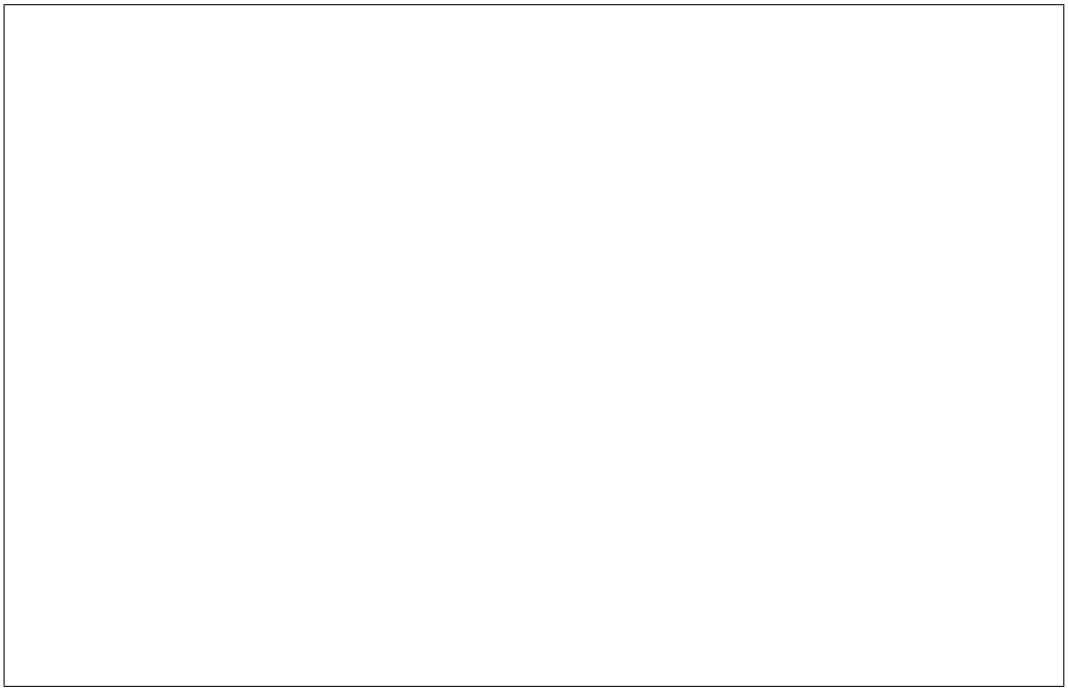
3. (question bonus) On suppose avoir la fonction `premierSuivant(n)` qui renvoie le premier nombre premier strictement supérieur à `n`. Par exemple, `premierSuivant(2)` vaut 3, `premierSuivant(3)` et `premierSuivant(4)` valent 5. (Il n'est pas demandé d'écrire cette fonction.)

   En utilisant la fonction `premierSuivant(n)`, écrire une fonction `decompositionFacteursPremiers(n)` qui renvoie la liste des nombres constituant la décomposition de `n` en un produit de facteurs premiers.

   Cette décomposition est obtenue en divisant `n` par 2 autant de fois que possible, en continuant de la même façon avec 3, puis avec 5, avec 7, et ainsi de suite jusqu'à ce que `n` ait la valeur 1.

   Par exemple, `decompositionFacteursPremier(504)` retournera la liste `[2,2,2,3,3,7]`, puisque $504 = 2 * 252$, $252 = 2 * 126$, $126 = 2 * 63$, et 63 est impair. Ensuite, $63 = 3 * 21$, $21 = 3 * 7$ et 7 n'est plus divisible par 3. Comme 7 n'est pas divisible par 5, 5 n'apparaît pas dans la liste. Enfin, $7 = 7 * 1$, il n'y a plus rien à décomposer, donc la liste est complète.

   Attention, inutile de chercher à utiliser les fonctions des questions précédentes.



Annexe : voici un rappel des principales fonctions disponibles pour manipuler les images et les graphes (à vous de voir celles qui sont utiles pour ce devoir).

L'argument `img` est une image	
`open(nom)`	Ouvre le fichier *nom* et retourne l'image contenue dedans (par exemple `open("teapot.png")`.
`Image.save(img, nom)`	Sauvegarde l'image *img* dans le fichier *nom*.
`new("RGB",(largeur,hauteur))`	Retourne une image de taille *large* × *haut*, initialement noire.
`(largeur,hauteur) = img.size`	Récupère la largeur et la hauteur de *img*.
`Image.putpixel(img, (x,y), (r,g,b))`	Peint le pixel (x, y) dans l'image *img* de la couleur (r, g, b)
`Image.getpixel(img, (x,y))`	Retourne la couleur du pixel (x, y) dans l'image *img*

L'argument `G` est un graphe	
`listeSommets(G)`	retourne la *liste* des *sommets* de `G`
`nbSommets(G)`	retourne le *nombre* de *sommets* de `G`
`sommetNom(G,etiquette)`	retourne le *sommet* de `G` désigné par son *nom* (`etiquette`). Exemple : `sommetNom(Europe, 'Italie')`

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