
A LIRE IMPERATIVEMENT AVANT DE COMMENCER LE TP

Consignes :

- au début du TP noté, créez un fichier de code source de la forme suivante : "NOM_Prenom.py", où votre nom est en majuscules et votre prénom commence par une majuscule, et sans aucun caractère accentué. C'est dans ce fichier de code source que vous allez travailler.
- les ressources du cours sont accessibles à l'adresse suivante : `http://dept-info.labri.fr/ENSEIGNEMENT/INITINFO/initinfo/support.html`
- faites régulièrement des sauvegardes de votre fichier de code source.
- les exercices sont indépendants et peuvent être traités dans l'ordre que vous voulez. Le sujet est volontairement long, le barême en tiendra compte. Attention cependant à ne pas aller trop vite, faites proprement ce que vous pouvez.
- pour vérifier que vos fonctions sont correctes, exécutez plusieurs tests. Ensuite, laissez au moins deux tests en commentaire (rappel : il suffit de mettre un # au début d'une ligne pour la commenter). L'absence de ces deux tests sera pénalisée.
- respectez le nom des fonctions demandées.
- à la fin du TP, envoyez un mail à "antonin.lentz@labri.fr" avec votre fichier de code source en pièce jointe. Mettez-vous en copie ("Cc :") et vérifiez que le courriel que vous recevez de vous-même contient bien votre fichier de code source en pièce jointe, et qu'il n'est pas vide.
- ne quittez pas la salle avant que votre responsable de groupe ne vous ait confirmé avoir reçu votre courriel, et que vous ayez émargé.

Exercice 1 – *Suites*

1. On définit la suite p_n comme suit : $\begin{cases} p_0 = 1 \\ p_{n+1} = p_n + \frac{(-1)^n}{2n+1} \quad \text{pour } n \geq 1 \end{cases}$
   Définir une fonction `suiteMystere(n)` qui renvoie p_n.
   Bonus : en commentaire, dites ce vers quoi semble converger la suite $(4p_n)_{n\in\mathbb{N}}$.
2. Pour un paramètre C entier strictement positif, on définit la suite de Syracuse par :
   - $u_0 = C$
   - $u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$

   On suppose que pour tout $C \geq 1$, il existe $n \geq 0$ tel que $u_n = 1$. Définir une fonction `syracuse(C)` qui renvoie le premier rang n tel que $u_n = 1$ (donc pour tout $k < n$, $u_k \neq 1$). On remarquera qu'après avoir atteint 1, la suite boucle sur les valeurs 4, 2 et 1. Après le TP, vous pourrez vous demander pourquoi un tel rang existerait toujours ?

Exercice 2 – *Couleur d'image*

1. Chaque pixel a une couleur contenant une composante rouge, verte et bleue. Définir une fonction `moyenneVert(img)` qui renvoie la valeur moyenne de la composante verte sur l'ensemble des pixels de l'image.
2. Définir une fonction `couleurMax(img)` qui modifie la couleur de chaque pixel de la façon suivante : on met à 0 les composantes étant strictement plus petites que les autres. Par exemple, si un pixel a pour couleur $(10, 35, 27)$, on la remplace par $(0, 35, 0)$. Mais si la couleur est $(78, 43, 78)$, on la remplace par $(78, 0, 78)$. C'est à dire qu'on ne conserve que la/les composante(s) maximale(s).

3. Définir une fonction `ligneMemeCouleur(img)` qui renvoie *True* si il existe une ligne de l'image `img` sur laquelle tous les pixels ont la même couleur, *False* sinon.

Exercice 3 – *Nombres premiers*

On rappelle qu'un nombre premier est un entier positif supérieur ou égal à 2, divisible uniquement par 1 et lui-même. Dans les fonctions suivantes, on supposera que l'argument `n` est un entier supérieur ou égal à 2.

1. Définir une fonction `estPremier(n)` qui renvoie *True* si `n` est premier, *False* sinon.
2. Définir une fonction `premierSuivant(n)` qui renvoie le plus petit nombre premier strictement supérieur à `n`. Par exemple, `premierSuivant(4)` renvoie 5 et `premierSuivant(7)` renvoie 11.
3. On admet que tout nombre entier strictement positif `n` peut s'écrire comme un produit de nombres premiers. Par exemple, $198 = 2 \cdot 3 \cdot 3 \cdot 11$. On admet aussi que cette décomposition est unique, à l'ordre près. Définir une fonction `decomposition(n)` qui affiche la décomposition en nombres premiers de `n`. Attention, si `n` = 198, il faut afficher deux fois 3 car il apparaît deux fois dans la décomposition.

Exercice 4 – *Graphes*

1. Définir une fonction `nbOccurences(L,x)` qui renvoie le nombre d'occurences de `x` dans `L`.
2. Définir une fonction `nbMultiAretes(G)` qui renvoie le nombre de multi-arêtes du graphe `G`.
3. Définir une fonction `accessibleMulti(G,s)` qui marque tous les sommets accessibles à partir du sommet `s` uniquement par des multi-arêtes. C'est à dire que pour passer d'un sommet à l'un de ses voisins, il faut qu'il existe au moins deux arêtes entre les deux sommets.
4. Définir une fonction `nbSommetsHautDegre(G,k)` qui renvoie le nombre de sommets du graphe `G` qui sont de degrés supérieurs ou égaux à `k`, ou ayant un voisin de degré supérieur ou égal à `k`.
5. Définir une fonction `nbSommetsHautDegreStrict(G,k)` qui renvoie le nombre de sommets du graphe `G` qui sont de degrés strictement supérieurs à `k`, ou ayant un voisin de degré strictement supérieur à `k`.