TP6 : Complexité

Démarrer le logiciel Spyder puis, dans le menu « Fichier » créer un nouveau fichier puis le sauvegarder avec un nom sous la forme TP6_votrenom. Dans la suite, on sauvegardera régulièrement son travail.

I Nombres premiers

Exercice 1 *Diviseurs*

1. Créer une fonction `diviseurs(n)` qui, pour un entier n donné, renvoie la liste de ses diviseurs positifs. Cette fonction a déjà été faite en cours.
2. Déterminer la complexité de cette fonction.
3. Écrire une fonction `nombres_diviseurs(n)` qui renvoie le nombre de diviseurs d'un entier n.
4. Écrire une fonction `produit_diviseurs(n)` qui renvoie le carré du produit des diviseurs d'un entier n qu'on notera $P(n)$.
5. En notant $d(n)$ le nombre des diviseurs de n, représenter sur un même graphique les suites $(n^{d(n)})$ et $P(n)$. Que remarque-t-on ?
6. *Question facultative :* Montrer la conjecture précédente mathématiquement.

Exercice 2 *Test de primalité et liste*

1. Créer une fonction `est_premier(n)` qui, pour un entier n donné, renvoie `True` si le nombre est premier et `False` sinon.
2. Créer une fonction `liste_premier(x)` qui renvoie la liste de tous les nombres premiers inférieurs à un flottant x donné.

Exercice 3 *Théorème des nombres premiers*

On va considérer la fonction la fonction $x \mapsto \pi(x)$ qui donne le nombre de nombres premiers inférieur ou égaux à x.
Par exemple $\pi(1) = 0$, $\pi(3.4) = 2$, $\pi(4) = 2$.

1. Créer une liste `liste_finie_premiers` qui contient tous les nombres premiers inférieurs à 10000.
2. Écrire une fonction `fonction_pi(x)` qui renvoie $\pi(x)$ lorsque $x \leqslant 10000$. *On utilisera la liste précédente pour éviter des lenteurs de calcul.*
3. Représenter sur un même graphique pour x allant de 1 à 100 puis 1000 puis éventuellement 9000, la fonction π ainsi que $x \mapsto \frac{x}{\ln x}$.
4. Interpréter le graphique précédent. *Ce théorème s'appelle le théorème de La Vallée-Poussin et est délicat à démontrer.*

II PGCD et algorithme d'Euclide

Exercice 4 *Fonction naïve*

1. Revoir le TP4 pour se souvenir de l'utilisation de `set` et `.intersection`.
2. Avec la fonction `diviseurs` précédente, écrire une fonction `PGCD_naif(a,b)` qui renvoie le PGCD de a et b.

Exercice 5 *Algorithme d'Euclide*

1. Revoir la fonction `bin` du chapitre 5.
2. On rappelle que pour calculer un PGCD de deux nombres a et b, on effectue des divisions euclidiennes successives et qu'on obtient le PGCD en prenant l'avant dernier reste. Programmer une fonction `PGCD_Euclide(a,b)` qui, avec ce procédé, renvoie le PGCD de a et b.
3. Tester pour plusieurs entiers $n \in \mathbb{N}$, `PGCD_Euclide(n+1,n)`. Que constate-t-on ?
4. Démontrer ce qui a été constaté à la question précédente.

Exercice 6 *Comparaison de complexité*

1. Écrire une fonction `comparaison(a,b)` qui pour deux entiers a, b renvoie une liste avec le temps de calcul de leur PGCD avec la fonction `PGCD_naif(a,b)` et `PGCD_Euclide(a,b)`.
2. Quelle fonction est-il préférable d'utiliser ?

III Développements limités

Exercice 7 *xlim,ylim*

Charger les modules `numpy` et `matplotlib.pypl` sous les noms `np` et `pypl` puis tester le code suivant :

```
def graph():
     abscisse = np.linspace(0, 2*pi, 30)
     ordonnee = [cos(x) for x in abscisse]
     pypl.xlim(-1,5)
     pypl.ylim(-4,8)
     return pypl.plot(abscisse,ordonnee)
```

Expliquer le fonctionnement de `xlim` et `ylim`.

Exercice 8 *Polynômes et graphiques*

1. Écrire une fonction Python `evalf_poly(a,b,c,x)` qui retourne, pour des flottants a, b, c, x :
$$P(x) = ax^2 + bx + c.$$
2. Écrire une fonction Python `graph_poly(n)` qui trace dans une fenêtre où les abscisses varient de -10 à 10 et les ordonnées de -10 à 10 les graphiques des fonction
$$P_k : x \mapsto \frac{1}{k}x^2 + 2x + 1$$
pour k variant de 1 à n.
3. Écrire une fonction Python `evalf_poly(L,x)` qui à partir d'une liste $[a_0, a_1, \ldots, a_n]$ renvoie
$$P(x) = a_0 + a_1 x + \cdots + a_n x^n.$$

Exercice 9 *Approximation de la fonction* sin

1. Rappeler le développement limité de la fonction sin à l'ordre n en 0.
2. Écrire une fonction Python `coeff_DL(n)` qui renvoie une liste avec les coefficients du développement limité de la fonction sin à l'ordre n en 0. Par exemple, pour $n = 3$, cette fonction retourne
`[1,0,-0.166666667`
car les premiers coefficients du développements limité de sin son 1, 0 et $-\frac{1}{3!} \simeq -0.16666667$. On pourra commencer par programmer, à nouveau, la fonction `facto`.
3. Écrire une fonction Python `approx(n)` qui trace sur un même graphique la fonction sin et la partie régulière du développement limité de sin à l'ordre n dans une fenêtre où les abscisses varient de -10 à 10 et les ordonnées de -10 à 10.

IV Calendrier

Exercice 10 *Années bissextiles*

On rappelle qu'une année est bissextile (comporte 366 jours au lieu de 365) lorsque

1. l'année est divisible par 4 et non divisible par 100 ;
2. l'année est divisible par 400.

Écrire une fonction Python `bissextile(n)` qui renvoie `True` si l'année est bissextile et `False` sinon.

Exercice 11 *Premier janvier et jour*

1. Sachant que le premier janvier 2013 est tombé un dimanche, écrire une fonction `jour(n)` retournant le jour dans la semaine où tombe le premier janvier d'une année. On prendra comme convention : 1 pour lundi, 2 pour mardi... jusqu'à 7 pour dimanche. Vérifier avec les exemples suivants : le premier janvier de 2042 tombe un dimanche et le premier janvier 2000 est tombé un samedi.
2. Écrire une fonction prenant en entrée une date (triplet d'entiers jour/mois/année) et retournant le jour dans la semaine correspondant à cette date.

Exercice 12 *Age*

Écrire une fonction Python qui prend en entrée deux liste à trois éléments de la forme `[annee,mois,jour]` (une pour la date du jour et une pour la date de naissance) et qui renvoie l'âge d'une personne.

V Compléments

Exercice 13 *Famille de polynômes*

On pose $P_0 = 2$, $P_1 = X$ et pour tout $n \in \mathbb{N}$,

$$P_{n+2}(X) = XP_{n+1}(X) - P_n(X).$$

1. Écrire une fonction Python `liste_coeff(n)` qui renvoie la liste des coefficients de P_n.
2. Représenter, sur un même graphique $P_0, P_1, \ldots, P_{10}$.
3. Montrer que pour tout $n \in \mathbb{N}$ et $\theta \in \mathbb{R}$,

$$P_n(2\cos\theta) = 2\cos(n\theta).$$

4. Déterminer les racines de P_n (on confirmera avec le graphique précédent).

Exercice 14 *Flocon de Von Koch*

On introduit d'abord un outil mathématique. On appelle similitudes directe du plan complexe les composées d'homothéties, de rotations et de translations. Si M est un point du plan d'affixe z, son image par une similitude a pour affixe

$$f(z) = az + b$$

où $a, b \in \mathbb{C}$ avec $a \neq 0$ caractérisent la similitude de la façon suivante :

1. si $a = 1$, la fonction f représente la translation de vecteur d'affixe b ;
2. si $a \notin \{0, 1\}$, en posant $a = \rho e^{i\theta}$ et en notant ω l'unique complexe qui vérifie $f(\omega) = \omega$), la similitude représentée par f est la composée d'une rotation de centre Ω d'affixe ω et d'angle θ et d'une homothétie de rapport ρ.

On note :

- f_1 la similitude de centre O de coordonnées $(0, 0)$, de rapport $\frac{1}{3}$ et d'angle 0 ;
- f_2 la similitude qui transforme O en B de coordonnées $\left(\frac{1}{3}, 0\right)$ et A de coordonnées $(1, 0)$ en C de coordonnées $\left(\frac{1}{2}, \frac{1}{2\sqrt{3}}\right)$;
- f_3 la similitude qui transforme O en C et A en D de coordonnées $\left(\frac{2}{3}, 0\right)$;
- f_1 la similitude de centre A de coordonnées $(0, 0)$, de rapport $\frac{1}{3}$ et d'angle 0.

1. Calculer les fonctions f_1, f_2, f_3 et f_4.
2. Charger le module `cmath` et définir les fonctions précédentes en Python.
3. Définir une fonction `successeur(L)` où, pour une liste de nombre complexe `L = [a1, ...,an]` renvoie `[f_1(a1), ...,f_1(an),f_2(a1), ...,f_2(an),f_3(a1), ...,f_3(an),f_4(a1), ...,f_4(an)]`.
4. Créer une fonction Python `iteration(n)` qui, à partir de la liste `[0,1]` effectue n fois l'étape précédente.
5. Écrire une fonction Python `fractale(n)` qui représente une ligne polygonale qui joint les points de la liste précédente.
6. Écrire finalement une fonction Python `flocon(n)` qui permet d'obtenir la figure ci-contre :