Lycée Loges

Chapitre 1
Algorithmie TP

Simon Dauguet
simon.dauguet@gmail.com

5 septembre 2024

Exercice 1 :
Écrire une fonction `discriminant(a:float, b:float, c:float) -> int` qui calcule le discriminant ($\Delta = b^2 - 4ac$) d'une équation polynomiale du 2ⁿ degré $ax^2 + bx + c = 0$ et précise le nombre de solutions réelles (0,1 ou 2).

Exercice 2 :
Faire une fonction `triG(n:int) -> None` qui affiche un triangle d'étoiles alignées à gauche sur n lignes :

```
*
**
***
****
```

Faire une fonction `triD(n:int) -> None` qui affiche cette fois un triangle d'étoiles alignés à droites sur n lignes :

```
   *
  **
 ***
****
```

Enfin, faire une fonction `losange(n:int) -> None` qui affiche un losange de "n étages" : avec $n = 4$, on aurait

```
---*---
--*-*--
-*---*-
*-----*
-*---*-
--*-*--
---*---
```

Exercice 3 :
Proposer une fonction `jeuProduit() -> None` qui génère deux chiffres aléatoires et demande à l'utilisateur la valeur du produit tant que la réponse est fausse.

Comment renvoyer le nombre d'essais avant réussite ? Changer le code de la fonction afin que soit affiché ce nombre d'essais.

Pour générer les deux entiers dont on cherche le produit, on utilisera les commandes

```
import random as rd
rd.randint(1,10000)
```

qui permet de générer un entier aléatoire entre 1 et 10000.

Exercice 4 (Le juste Prix) :
On souhaite jouer au jeu du "juste prix". Le maître du jeu (ou l'ordinateur) choisi un entier entre 0 et 100 et le joueur doit trouver cet entier avec le moins de coup possible. Selon sa réponse, le maître du jeu dit si le nombre recherché est plus petit, plus grand ou égal à sa proposition.

Faire une fonction `justePrix() -> None` qui permette de jouer.

La fonction `randint(a,b)` dans le module `random` permet de générer un nombre aléatoire entre a et b.

Exercice 5 (Conjecture de Syracuse) :

On considère la suite de Syracuse (u_n) définie par $u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ est impair} \\ \frac{u_n}{2} & \text{si } u_n \text{ est pair} \end{cases}$.

1. Vérifier que les valeurs 4, 2, 1 forment un cycle.
2. Proposer une fonction `syracuse(u0 : int) -> list` qui calcule et renvoie la liste de chaque terme u_n jusqu'à obtenir 1.
3. Modifier le code précédent afin de renvoyer la longueur du vol (*i.e.* premier n tq $u_n = 1$).

Exercice 6 :
Soit `tab` un tableau de nombres. Proposer une fonction `sommePondere(tab:list) -> float` qui calcule la somme des éléments de `tab` en multipliant chacun des nombres par sa position (donc en pondérant le nombre de son indice). Exemple : si $tab = [3, 2, 5]$, on doit obtenir $3 \times 0 + 2 \times 1 + 5 \times 2 = 12$.

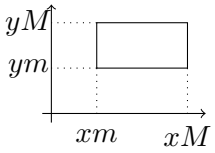
Exercice 7 :
Soit `tab` un tableau de n nombres, proposer une fonction `diffSuccessive(tab:list) -> list` qui créer un nouveau tableau de taille $n - 1$ contenant la différence d'une case avec sa suivante. Exemple : si $tab = [3, 2, 5]$, on doit obtenir $[1, -3]$.

Exercice 8 :
Proposer une procédure `convertime(nbs : int) -> None`, qui prend en argument un entier `nbs` représentant un nombre de secondes et affiche la conversion en jours, heure, minutes et secondes.

Exercice 9 :
Écrire une fonction `presque_egal(tab : list, val : float) -> list`, qui renvoie un tableau contenant tous les éléments de `tab` de valeur comprise entre `val-1` et `val+1`.

Exercice 10 :

On défini un rectangle par les coordonnées de ses côtés `(xm,xM,ym,yM)`.
Écrire une fonction `intersecRect(rect1 : tuple, rect2 : tuple) -> bool`, qui renvoie `True` si les deux rectangles s'intersectent, `False` sinon.
Puis écrire une fonction `rectIntersec(rect1 : tuple, rect2 : tuple) -> tuple`, qui renvoie les coordonnées du rectangle d'intersection ou `None` sinon.



Exercice 11 :

Écrire une fonction `melangeAleaT(t1 : list, t2 : list) -> list`, qui crée un nouveau tableau de la taille du plus grand tableau avec des éléments de `t1` et `t2` choisis aléatoirement. Exemple : si `t1=[1,3,3]` et `t2=[2,2,4,6]`, on pourra obtenir `[1,3,2,1]`.

On pourra utiliser la fonction `randint(a,b)` qui choisit un entier aléatoire dans $[a, b]$ dans le package `random`.

Exercice 12 :

On souhaite calculer la variance d'une série de valeurs contenues dans un tableau.

1. Proposer une fonction `moyenne(tab : list) > float`, qui renvoie la moyenne des valeurs contenues dans `tab`
2. Écrire une fonction `diffCarre(x : float, m : float) -> float`, qui renvoie $(x - m)^2$.
3. On rappelle la définition de la variance : la moyenne des carrés des écarts à la moyenne. Finaliser la fonction `variance(tab : list) -> float`.

Exercice 13 (Test de primalité) :

Écrire une fonction `isPrime(n:int) -> bool` qui prend en argument un entier n et qui renvoie `True` ou `False` selon si n est premier ou non.

Exercice 14 (Entiers premiers jumeaux ; Variables globales) :

Le but de cet exercice est de donner une liste de nombres premiers jumeaux compris entre deux bornes. Pour faciliter les choses, on pourra utiliser deux variables globales `premiers` et `maxp`, la première contenant la liste des nombres premiers et la seconde contenant la valeur du plus grand premier trouvé pour le moment (donc `maxp` est le maximum de `premiers`). On débutera avec `premiers=[2,3]` et `maxp=3`.

1. Écrire une fonction `divisible(n:int) -> bool` prenant en argument un entier $n \geq 2$ et renvoyant `True` si n est divisible par au moins un nombre premier de la liste `premiers`, et `False` sinon.
2. Écrire une fonction `trouve_premier(n:int) -> None` qui actualise la liste `premiers` pour qu'elle contienne la liste de tous les nombres premiers inférieurs ou égaux à n (et qui actualise aussi `maxp` par la même occasion).
3. On dit que deux nombres premiers sont jumeaux s'ils s'écrivent p et $p + 2$. Une conjecture (toujours non démontrée à l'heure actuelle) affirme qu'il en existe une infinité (on a même une idée sur leur répartition en $x/\ln(x)^2$, mais cette répartition est aussi conjecturale (voir image.math.cnrs)). Écrire une fonction `jumeaux(n:int, m:int) -> list` qui prend en argument deux entiers $n \leq m$ et qui renvoie la liste des couples $(p, p + 2)$ des nombres premiers jumeaux avec $n \leq p \leq m$.

Exercice 15 (Entier premier de Limerick) :

Un limerick est un poème anglais humoristique formé de 5 vers rimés de la forme $AABBA$.

Faire une fonction `Limerick() -> list` qui renvoie la liste de tous les nombres premiers de la forme $aabba$ (par exemple, 11221 ne doit pas faire partie de la liste).

Exercice 16 (Décomposition d'entiers) :
Quel est le plus grand entier naturel non nul inférieur à 1000 que l'on ne peut écrire sous la forme $13a+17b+19c$ avec $a, b, c \in \mathbb{N}$?

Exercice 17 (Somme de cubes) :
Écrire une fonction qui renvoie le premier entier naturel inférieur à 10000 qui peut s'écrire de deux façons différentes comme la somme de deux cubes.

Exercice 18 (Théorème d'Erdòs-Suranyi) :
Le théorème d'Erdòs-Suranyi affirme :

$$\forall n \in \mathbb{N},\ \exists p \in \mathbb{N},\ n = \sum_{k=1}^{p} \varepsilon_k k^2$$

où $\forall k \in \mathbb{N}$, $\varepsilon_k \in \{-1, 1\}$.

Écrire une fonction `Erdos(n:int) -> list` qui renvoie la liste des entiers qui décompose n avec les signes. Par exemple, `Erdos(6)` devra renvoyer `[[1,1],[-1,2],[1,3]]` car $6 = 1^2 - 2^2 + 3^2$.

Exercice 19 (Décomposition de Zeckendorf) :
On rappelle que la suite de Fibonacci est définie par

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \in \mathbb{N},\ F_{n+2} = F_n + F_{n+1} \end{cases}$$

Le théorème de Zeckendorf établit que n'importe quel entier naturel peut s'écrire de façon unique comme une somme de termes distincts de la suite de Fibonacci de rang supérieur à 2 et non consécutifs :

$$\forall n \in \mathbb{N},\ \exists! p \in \mathbb{N}^*,\ \exists!(\alpha_0, \ldots, \alpha_p) \in \mathbb{N}^{p+1},\ \alpha_0 \geq 2,\ \forall i \in \{0, \ldots, p-1\},\ \alpha_{i+1} > \alpha_i + 1,$$
$$n = \textstyle\sum_{k=0}^{p} F_{\alpha_k}.$$

Par exemple, $10 = 8 + 2 = F_6 + F_3$ et $8 = F_6$. Mais $8 = F_5 + F_4$ n'est pas la décomposition de Zeckendorf de 8. Ni $8 = F_5 + F_3 + F_2$, ni $8 = F_5 + F_3 + F_1$ car l'indice de F_1 n'est pas supérieur à 2. On peut aussi écrire $100 = F_{11} + F_6 + F_4 = F_{11} + F_6 + F_3 + F_2 = F_{10} + F_9 + F_6 + F_4$ mais seule la première décomposition convient.

Faire une fonction `zeckendorf(n:int) -> list` prenant en argument un entier naturel et renvoyant la liste des indices des termes de la suite de Fibonacci qui le compose par la décomposition de Zeckendorf. Donc la fonction devra renvoyer `[6,3]` pour 10, `[6]` pour 8 et `[11,6,4]` pour 100.