Lycée Loges

Chapitre 1

Algorithmie Exercices

Simon Dauguet
simon.dauguet@gmail.com

5 septembre 2024

1 Entrée / sortie

Exercice 1 :
On considère une valeur x connue. Afin de calculer l'image x^3+x^2+x on a besoin de 5 opérations ($x^3 = x \times x \times x$ correspond à 2 opérations). Proposer un calcul qui requiert moins d'opérations (il s'agit de la méthode de Ruffini-Horner).

Exercice 2 :
Proposer une fonction Python `aire(l:float) -> float` qui demande la longueur d'un côté d'un carré et renvoie son aire.

Exercice 3 :
En s'inspirant de l'exercice précédent, on souhaite créer une fonction `age() -> int` qui demande l'âge de l'utilisateur et calcule son âge en 2050.

Exercice 4 :
Écrire une procédure `nbJour(J:int) -> None` qui demande un nombre de jours et affiche la conversion en année (composée de 365j) et jours restants.

Exercice 5 :
Écrire une fonction `temps(N:int) -> list` qui demande un nombre de secondes et effectue la conversion en jours, heures, minutes et secondes et renvoie la liste des jours, heures, minutes correspondante.

2 Tests conditionnels

Exercice 6 :
Proposer une fonction `estPair(n:int) -> bool` qui précise si le nombre `n` est pair ou non.

Exercice 7 :
Proposer une procédure `greetings() -> None` qui demande l'âge de l'utilisateur et affiche "Bonjour" s'il a plus de 18 ans, "Salut" sinon.

Exercice 8 :
Proposer une procédure `jeu() -> None` qui demande un nombre et qui vérifie si ce nombre est compris entre 0 et 20. S'il est à l'extérieur, il affiche un message d'erreur. S'il est supérieur à 15, il affiche "Bravo !".

Exercice 9 :
Proposer une fonction `parabole(M:tuple) -> bool` qui vérifie si le point $M(x;y)$ appartient à la parabole d'équation $y = x^2$.

Exercice 10 :
Proposer une fonction `aireParallelogramme(u:tuple, v:tuple) -> float` qui renvoie l'aire du parallélogramme définit par les deux vecteurs `u` et `v` (dans $\mathbb{R}^2$).

Exercice 11 :
On considère trois nombres réels x, y et z. Proposer une fonction `ordre(x:float;, y:float, z:float) -> tuple` qui permettent d'ordonner ces 3 nombres. On essaiera de minimiser le nombre de comparaisons.

3 Boucles finies

Exercice 12 :
Écrire une procédure `table7(n:int) -> None` qui affiche la table de multiplication de 7 jusqu'à n. Puis faire une nouvelle fonction `table(p:int, n:int) -> None` qui fait de même avec la table de `p`.

Exercice 13 :
Écrire une fonction `pairs(n:int, m:int) -> list` qui renvoie les nombres pairs entre n et m compris.

Exercice 14 :
On considère la fonction f définie par $f(x) = x^2 + 2$. Proposer une fonction Python `imagef(n:int) -> list` qui renvoie la liste des images par f des entiers compris entre $-n$ et n.

Exercice 15 :
Une personne a mis 1000€ dans un compte rémunéré à 2% mensuel. Combien y a-t-il au bout d'un an ? 2 ? 3 ?

Proposer une fonction python `interet(n:int) -> flaot` qui calcule la somme d'argent disponible au bout de n années.

Exercice 16 :
Écrire une fonction `carres(n:int) -> list` qui renvoie la liste de tous les carrés des nombres entiers inférieurs à n (sans utiliser de boucle `while`).

Exercice 17 :
Proposer une fonction `diviseur(n:int) -> list` qui renvoie la liste tous les diviseurs positifs de n.

Exercice 18 :
Écrire une fonction `puissance(x:float, n:int) -> float` qui calcule par multiplication successive x^n.

Exercice 19 (retour sur Ruffini-Horner) :
On souhaite calculer $S_n = \sum_{k=0}^{n} x^k$. On pose u_n le nombre d'opérations faites lors du calcul naïf et v_n avec la méthode de Ruffini-Horner.

1. déterminer les valeurs de u_n et v_n pour $n \in [\![1;4]\!]$
2. définir alors les suites (u_n) et (v_n) pour $n \in \mathbb{N}^*$
3. proposer une fonction python `somme(x:float, n:int) -> tuple` qui, sachant un nombre x et un entier n, affiche la valeur de la somme S_n ainsi que le nombre d'opérations faites.

Exercice 20 :
À l'aide d'une boucle `for`, écrire une fonction python `sommeab(a:int, b:int) -> int` qui calcule la somme des termes compris entre deux entiers a et b.

N'existerait-il pas une formule mathématique pour cela pour éviter la boucle ?

Exercice 21 (Factorielle) :
Proposer une fonction `factorielle(n:int) -> int` qui renvoie la valeur $n!$.

Exercice 22 :
Soit (f_n) la suite de Fibonacci définie par $f_0 = 0$, $f_1 = 1$ et $f_{n+1} = f_n + f_{n-1}$ pour $n \geq 1$.

1. Calculer les valeurs de f_n pour $n \in [\![2;5]\!]$
2. Proposer une fonction `fibonacci(n:int) -> int` qui renvoie f_n.

4 Boucles conditionnelles

Exercice 23 :
On considère la suite u définie par $\begin{cases} u_0 = 3 \\ u_{n+1} = 3u_n - 4 \end{cases}$.

1. Démontrer que $u_n = 3^n + 2$ pour tout entier naturel n.
2. Proposer une fonction `seuilInf(A:float) -> float` qui renvoie le premier indice dont le terme est supérieur à A.

Exercice 24 :
On considère la suite u définie par $\begin{cases} u_0 = 1 + u_1 = 2 \\ u_{n+2} = 2u_{n+1} - u_n \end{cases}$.

1. Démontrer que $\forall n \in \mathbb{N}$, $u_n = 2 - n$.
2. Proposer une fonction `seuilNeg(N:int) -> int` qui renvoie le premier indice dont le terme est inférieur à $N < 0$.

Exercice 25 :
Proposer une fonction `moyenne() -> float` qui demande à l'utilisateur le nombre n de notes qu'il a, puis toutes ses notes et lui renvoie sa moyenne.

Exercice 26 :

1. Proposer une fonction `double(x:float, lim:int) -> float` qui doit doubler la valeur de `x` jusqu'à dépasser la valeur de `lim`. La fonction renverra cette valeur de `x`.
2. Modifier votre code afin qu'il compte le nombre d'étapes effectuées.
3. Application : une feuille de papier possède une épaisseur de 0,1mm et la distance terre-lune est d'environ 380 000km. Combien de fois selon vous, faut-il plier la feuille de papier pour dépasser la distance terre-lune ? Faites tourner votre programme et comparer.

5 Listes et chaînes de caractères

Exercice 27 :
On considère la variable `mot` de type `str`.

1. Quelle commande python permet de construire la sous-chaîne de caractères contenant les lettres de `mot` de 2 en 2 (la 1è lettre, puis la 3è, etc.)
2. Et si l'on veut de la dernière lettre à la première ?

Exercice 28 :
On considère la variable `mot` de type `str`, proposer une fonction `contientA(mot:str) -> bool` qui vérifie à l'aide d'une boucle l'existence de la lettre `"a"` dans la variable `mot`.

Exercice 29 :
On considère la variable `mot` composée de caractères minuscules. Proposer une fonction `alphabet(mot:str) -> list` qui crée un tableau de 26 cases comptant le nombre de chacune des lettres de `mot` (`'a'` étant à l'indice 0).

Exercice 30 :
Soit un tableau `tab`, quelle commande permet d'avoir le tableau `tab` "à l'envers" (en partant du dernier) ?

Exercice 31 :
Proposer une fonction `sommeT(tab:list) -> float` qui calcule la somme du tableau de nombres `tab`.

Exercice 32 :
Soit `tab` un tableau de nombres. Proposer une fonction `sommePuissance(tab:list) -> float` calcule la somme des éléments de `tab` en mettant chaque nombre à la puissance de sa position. Exemple : si $tab = [3, 2, 5]$, on doit obtenir $3^0 + 2^1 + 5^2 = 28$.

Exercice 33 :
À partir d'un tableau `tab`, proposer une fonction `superposition(tab:list) -> list` qui crée un nouveau

tableau et additionne le premier élément avec le dernier, le deuxième avec l'avant-dernier, etc. Exemple : si `tab=[1, 5, 3]`, on aura `[4, 10, 4]`.

Exercice 34 :
À partir d'un tableau `tab`, proposer une fonction `superpositionTronquee(tab:list) -> list` qui crée un demi-tableau composé de la somme 1er élément avec le dernier, le 2è avec l'avant-dernier, etc. Exemple : si `tab=[1, 5, 7, 3]`, on aura `[4, 12]`.

Exercice 35 :
À partir de deux tableaux `tab` et `tab2`, proposer une fonction `concat(tab:list, tab2:list) -> list` qui fasse "à la main" la concaténation des deux tableaux, *i.e.* qui crée un nouveau tableau et met bout à bout les deux tableaux avec une boucle `for` ou `while`.

Exercice 36 :
Soit `tab` un tableau de n nombres et $k \in [\![1; n]\!]$. Proposer une fonction `sommePlage(tab:list, k:int) -> list` qui créé un nouveau tableau contenant à l'indice i la somme des valeurs du tableau comprises entre les indices i et $i + k - 1$. Exemple : si `tab=[1, 3, 5, 7]` et `k=2`, on aura `[4, 8, 12]`.

6 Algorithmie complète

Exercice 37 :
Prédire l'affichage des codes suivants :

```
1 def f(a : int) -> None:
2     print(a+y)
3 y=2
4 a=5
5 f(4)
```

```
1 def f(a : int) -> None:
2     y=6
3     print(a+y)
4 y=2
5 f(4)
```

```
1 def f(a : int) -> None:
2     a=7
3     y=2
4     print(a+y)
5 f(4)
```

Exercice 38 :
Proposer une fonction `zerof(a : float, b : float, c : float) -> bool`, qui étant donné 3 réels a, b et c définissant la fonction polynomiale du 2n degré $x \mapsto ax^2 + bx + c$ renvoie un booléen selon s'il existe des racines réelles ou non.

Exercice 39 :
Proposer une fonction `fmax(a : float, b : float) -> float`, qui prend deux flottants a et b en argument, renvoie le plus grand flottant.

Exercice 40 :
Proposer une fonction `fabs(a : float) -> float`, qui prend un nombre `a` en argument et renvoie sa valeur absolue (sans utiliser la fonction `abs` évidemment).

Exercice 41 :

Proposer une fonction `fmax3(x : float, y : float, z : float) -> float`, qui prend en argument trois flottants et renvoie le plus grand (sans utiliser la fonction `max` évidemment).

Exercice 42 :
Proposer une fonction `isBissextile(an : int) -> bool`, qui prend en argument un entier `an` et renvoie un booléen selon que l'année est bissextile ou non.

Exercice 43 :
Proposer une fonction `puissance(x : float, n : int) -> float`, qui prend en argument un flottant x et un entier naturel n et renvoie x^n. On utilisera une boucle `for`/`while` à l'intérieur de la fonction.

Exercice 44 :
Écrire une fonction `positifsT(tab : list) -> bool`, qui renvoie `True` si tous les nombres du tableau `tab` sont positifs, `False` sinon.

Exercice 45 :
Écrire une fonction `nb_valeur_sup(tab : list, a : float) -> int`, qui renvoie le nombre d'éléments de `tab` supérieur à a. Puis faire de même avec entre a et b.

Exercice 46 :
Écrire une fonction `nbChiffres(n : int) -> int`, qui donne le nombre de chiffres qui compose l'écriture en base 10 de l'entier n.

Exercice 47 :
Écrire une fonction `element_plus_grand(tab : list, val : float) -> list`, qui renvoie un tableau contenant tous les éléments de `tab` plus grand que `val`.

Exercice 48 :
Écrire une fonction `occurrence(let : str, mot : str) -> int`, qui renvoie le nombre de fois que la lettre `let` est dans `mot`.

Exercice 49 :
Écrire une fonction `occurrenceT(val : float, tab : list) -> int`, qui renvoie le nombre de fois que la valeur `val` se trouve dans le tableau `tab`.

Exercice 50 :
Écrire une fonction `maxT(tab : list) -> float`, qui renvoie l'élément le plus grand du tableau `tab`, sans utiliser la fonction `max`.

Exercice 51 :

Même chose avec `maxT2(tab : list) -> float`, qui renvoie le 2è plus grand élément.

Exercice 52 :
Écrire une procédure `echangeT(tab : list, i : int, j : int) -> None`, qui échange le iè élément avec le jè si les indices i et j sont possibles.

Exercice 53 :

1. Définir une fonction `f(x : float) -> float`, définie par $f(x) = 4(x-2)^2 + 1$
2. Écrire alors une fonction `postTraitement(tab : list, fct : object) -> list`, qui applique la fonction `fct()` sur chaque élément du tableau `tab` pour créer un nouveau tableau qui sera renvoyé.

Exercice 54 :
On souhaite proposer un automate qui rend au mieux la monnaie à l'aide de coupures de 20, 10, 5, 2 et 1 euro.
Proposer une fonction `rendu2monnaie(diff : int) -> list`, qui renvoie un tableau contenant le nombre de coupures à rendre partant de 20€ à 1€. Exemple : `rendu2monnaie(49)` renverra `[2,0,1,2,0]`.

Exercice 55 (Loi de Benford) :
La loi de Benford est une loi empirique de probabilité qui dit que, dans une série de données chiffrées, il y a plus de nombres qui commencent par le chiffre 1 que par le chiffre 2, et il y a plus de nombres qui commencent par le chiffre 2 que par le chiffre 3, etc.

1. Faire une fonction `Alea(n:int) -> int` qui prend en paramètre un entier `n` et qui renvoie un entier composé de `n` chiffres choisit aléatoirement, de démarrant pas par 0.
2. Faire une fonction `Decompte(N:int) -> list` qui donne le nombre d'apparition du chiffre 1 dans le nombre N, le nombre d'apparition du chiffre 2 dans le nombre N etc.
3. Faire une fonction `Benford(n:int, N:int) -> tuple` qui test la loi de Benford pour N nombres de longueurs n. La fonction renverra un 9-uplet contenant les taux d'apparition de chaque chiffre en premier, en pourcentage.
4. Tester la loi de Benford sur 5 séries de 1000 entiers de longueurs 10. Qu'en pensez-vous ?

Remarque :
La loi de Benford a été utilisé pour détecter des fraudes fiscales, comptables ou même électorales.

Exercice 56 (Les carrés dans tous leurs états) :
Trouver tous les carrés inférieurs à N (pas trop trop gros) pouvant s'écrire comme la somme de nombres composés d'un même chiffre répété au moins deux fois. Par exemple, $11^2 = 99 + 22$ ou $38^2 = 333 + 1111$. On donnera le nombre qui convient (ici 11 et 38) ainsi que les nombres qui apparaissent dans la somme (donc 99 et 22 pour le premier et 333 et 1111 pour le second exemple).