

- Sans document, sans calculatrice ou autre dispositif électronique.
- Lorsque l'on demande d'écrire un code Python, on prendra soin de numéroté les lignes et de faire apparaître très clairement l'indentation.

**Exercice 1.** Déterminer la valeur retournée par chacun des appels demandés.

```
1 def f(x):  
2     if x % 2 == 0:  
3         return 3  
4     if x % 4 == 0:  
5         return 5  
6     else:  
7         return 7
```

1. f(3)

2. f(6)

3. f(8)

4. f(1)

**Exercice 2.** Déterminer la valeur retournée par chacun des appels demandés.

```
1 def f(x):  
2     if x % 2 == 0:  
3         a = 3  
4     if x % 4 == 0:  
5         a = 5  
6     else:  
7         a = 7  
8     return a
```

1. f(3)

2. f(6)

3. f(8)

4. f(1)

**Exercice 3.** Déterminer la valeur retournée par chacun des appels demandés.

```
1 def f(x):  
2     if x % 2 == 0:  
3         a = 3  
4     elif x % 4 == 0:  
5         a = 5  
6     else:  
7         a = 7  
8     return a
```

1. f(3)

2. f(6)

3. f(8)

4. f(1)

**Exercice 4.** Expliquer pourquoi les instructions suivantes renvoient des messages d'erreur.

1.

```
1 def f(x):  
2     a = 0  
3     if x % 2 == 0:  
4         a = 0  
5     else x % 4 == 0:  
6         a = 1  
7     return a
```

2.

```
1 def g(x):  
2     a = 0  
3     if x % 2 = 0:  
4         a = 0  
5     else:  
6         a = 1  
7     return a
```

**Exercice 5.** On donne la fonction `lapin`

```
1 def lapin(n):  
2     a = 0  
3     b = n + 1  
4     while a + 1 < b:  
5         x = (a + b) // 2  
6         if x * x <= n:  
7             a = x  
8         else:  
9             b = x  
10        print(a, b, x)  
11    return b - 1
```

1. Donner les valeurs affichées par l'instruction `print` lors de l'appel `lapin(52)`.
2. Quelle valeur retourne la fonction lors de l'appel `lapin(52)` ?
3. Donner les valeurs affichées par l'instruction `print` lors de l'appel `lapin(0)`.
4. Quelle valeur retourne la fonction lors de l'appel `lapin(0)` ?

**Exercice 6.** Si  $x$  et  $y$  sont deux nombres réels quelconques, on définit une fonction prenant comme variables  $x$  et  $y$  de la façon suivante

$$f(x, y) = \begin{cases} x^2 + y - 3 & \text{si } x > y \\ x^2 - y^2 + 12 & \text{si } x < y \\ x - 1 & \text{si } x = y. \end{cases}$$

1. Écrire une fonction `f` prenant comme paramètres deux flottants («nombres à virgule»)  $x$  et  $y$  et calculant  $f(x, y)$ .
2. On définit une suite  $(u_n)_{n \in \mathbb{N}}$  par  $u_0 = 0$  et la relation

$$\forall n \in \mathbb{N}, u_{n+1} = f(u_n^2, u_n).$$

Écrire une fonction `carotte` ayant pour argument un entier naturel  $n$  et qui retourne la valeur de  $u_n$ .

### Exercice 7.

1. Écrire une fonction `nb_chiffres` ayant pour arguments deux entiers  $n$  et  $b$  avec  $n \geq 1$  et  $b \geq 2$  et qui retourne l'entier  $p$  tel que

$$b^{p-1} \leq n < b^p.$$

On utilisera uniquement les opérations sur les entiers sans import de bibliothèque.

2. Écrire un ensemble de tests fonctionnels pour votre fonction `nb_chiffres`.

**Exercice 8.** Écrire une fonction `seconddegre` avec trois paramètres  $a$ ,  $b$ ,  $c$ , et qui renvoie la liste contenant les racines réelles du trinôme du second degré

$$aX^2 + bX + c$$

- Préconditions :  $a$ ,  $b$ ,  $c$  sont trois flottants et  $a$  est non nul.

On obtiendra par exemple

#### Python 3 Shell

```
>>> seconddegre(1, -5, 6)
[2.0, 3.0]
>>> seconddegre(1, -2, 1)
[1.0]
>>> seconddegre(1, 1, 1)
[]
```

**Exercice 9.** On donne trois nombre  $a, b, c$  et on souhaite les ordonner dans l'ordre croissant. Voici donc mon programme:

```
1 def tri(a, b, c):
2     """
3     Retourne la liste triée contenant les trois nombres a, b, c
4     """
5     if a > b:
6         x = a
7         a = b
8         b = x
9     if b > c:
10        x = b
11        b = c
12        c = x
13    if a > c:
14        x = a
15        a = c
16        c = x
17
18    return [a, b, c]
```

Je l'essaie avec  $a = 3, b = 1$  et  $c = 2$ .

#### Python 3 Shell

```
>>> tri(3, 1, 2)
[1, 2, 3]
```

1. Que font les lignes 5-6-7-8 dans le programme précédent ?
2. Écrire un ensemble de tests fonctionnels «crédible» `test_tri` pour ma fonction de tri.
3. Lesquels de vos tests échouent ? Expliquez ?

**Exercice 10.** Prenons un nombre entier naturel  $n$  non nul. Calculons la somme des carrés de ses chiffres. On obtient un nouveau nombre. Recommençons cette opération avec ce nouveau nombre, on obtient un troisième nombre.

Que se passe-t-il si on itère ces opérations plusieurs fois? Observons deux exemples.

- Avec  $n = 19$ , on obtient successivement

$$19 \rightarrow 82 \rightarrow 68 \rightarrow 100 \rightarrow 1 \rightarrow 1 \dots$$

et le 1 se répète indéfiniment.

- Avec  $n = 200$ , on obtient successivement

$$200 \rightarrow 4 \rightarrow 16 \rightarrow 37 \rightarrow 58 \rightarrow 89 \rightarrow 145 \rightarrow 42 \rightarrow 20 \rightarrow 4 \dots$$

et le cycle  $4 \rightarrow 16 \rightarrow 37 \rightarrow 58 \rightarrow 89 \rightarrow 145 \rightarrow 42 \rightarrow 20$  se répète indéfiniment.

On peut démontrer qu'en appliquant un tel processus, à partir d'un entier quelconque non nul, on finit toujours par arriver sur l'une ou l'autre de ces deux situations, c'est-à-dire à arriver sur le nombre 1 ou sur le nombre 4.

On dira qu'un nombre entier naturel non nul  $n$  est **heureux** si le processus mène à 1, et **malheureux** s'il mène à 4.

1. Quelles sont les opérateurs Python permettant d’obtenir le quotient et le reste de la division euclidienne d’un entier  $n$  par un autre entier  $p$ .
2. Écrire une fonction `somme_carres` qui renvoie la somme des carrés des chiffres de l’entier passé en paramètre.

#### Python 3 Shell

```
>>> somme_carres(19)
82
>>> somme_carres(200)
4
```

3. Après avoir pris soin de bien numéroter les lignes de votre fonction `somme_carres`, simuler l’exécution de l’appel `somme_carres(3407)`. On présentera l’évolution des valeurs prises par vos variable sous forme d’un tableau. Préciser à chaque fois le numéro de la ligne exécutée.
4. Réalisez un prédicat (c’est-à-dire une fonction qui retourne un booléen) nommé `est_heureux` qui renvoie `True` si l’entier passé en paramètre est heureux, et `False` sinon.

#### Python 3 Shell

```
>>> est_heureux(19)
True
>>> est_heureux(200)
False
```

5. On appelle **altitude**, le terme maximum atteint par la suite obtenue à partir de l’entier  $n$ . Par exemple, l’altitude de 19 est 100 et l’altitude de 200 est 200.  
Écrire une fonction `altitude` ayant pour paramètre un entier naturel  $n$  et qui retourne l’altitude partant de  $n$ .

#### Python 3 Shell

```
>>> altitude(19)
100
>>> altitude(200)
200
```

- Sans document, sans calculatrice ou autre dispositif électronique.
- Lorsque l'on demande d'écrire un code Python, on prendra soin de numéroté les lignes et de faire apparaître très clairement l'indentation.
- La complexité, ou le temps d'exécution, d'un programme P est le nombre d'opérations élémentaires nécessaires à l'exécution de P. Lorsqu'il est demandé de donner une certaine complexité, cette dernière sera donnée par une borne supérieure asymptotique (en  $O(\dots)$ ) et sera justifiée.

## Lecture de code

**Exercice 1.** Voici une fonction écrite en Python

```
1 def mystere(a):  
2     s = 0  
3     b = a  
4     while b > 0:  
5         s = s + b % 10  
6         b = b // 10  
7     return s
```

1. Exprimez en français ce que font respectivement les lignes 5 et 6.
2. Donnez, sous forme d'un tableau dont les colonnes représentent les variables s et b les valeurs successives de ces deux variables lors du calcul de `mystere(3209)`.
3. Lorsque a est un nombre entier positif ou nul, que représente la valeur de l'expression `mystere(a)`?
4. Quelle est la valeur de l'expression `mystere(a)` lorsque a est un entier négatif?
5. Comment est affectée la valeur retournée si l'on intervertit l'ordre des deux lignes 5 et 6 dans la fonction `mystere`?
6. Réécrivez la fonction `mystere` de sorte que l'expression `help(mystere)` donne une documentation utile au programmeur.
7. Écrire un jeu de tests adaptées à la fonction `mystere`.

**Exercice 2.** On considère la fonction `caroline` suivante ayant pour paramètre une liste de nombres `A`:

```
1 def caroline(A):
2     n = len(A)
3     for i in range(1, n):
4         j = i
5         while j > 0 and A[j - 1] > A[j]:
6             A[j - 1], A[j] = A[j], A[j - 1]
7             j = j - 1
8         print(i, j, A)
```

1. Donner la suite de lignes affichées par l'instruction `print(i, j, A)` lors de l'appel

**Python 3 Shell**

```
>>> caroline([5, 11, 2, 4, 1, 8])
```

2. Donner la suite de lignes affichées par l'instruction `print(i, j, A)` lors de l'appel

**Python 3 Shell**

```
>>> caroline([8, 5, 3, 1])
```

3. Que fait la fonction `caroline` ?
4. Déterminer la complexité en temps dans le pire cas de la fonction `caroline` en fonction de `n = len(A)`.
5. Déterminer la complexité en temps dans le meilleur cas de la fonction `caroline` en fonction de `n = len(A)`.

**Exercice 3.** On considère la fonction suivante qui calcule la racine carrée entière par défaut

```
1 def racine(n):
2     """
3     Donnée: n : nombre dont on veut calculer la racine (entier)
4     Résultat: r : racine carrée entière par défaut (entier >= 0)
5
6     Pré condition: n >= 0
7     Post condition: r * r <= n < (r + 1) * (r + 1)
8     """
9     r = 0
10    y = 1
11    z = 1
12    while y <= n:
13        z = z + 2
14        y = y + z
15        r = r + 1
16    return r
```

1. Démontrer la validité du triplet de Hoare suivant:

$$\begin{aligned} & \{ y \leq n \text{ et } y = (r + 1)^2 \text{ et } z = 2r + 1 \text{ et } r^2 \leq n \} \\ & z = z + 2 \\ & y = y + z \\ & r = r + 1 \\ & \{ y = (r + 1)^2 \text{ et } z = 2r + 1 \text{ et } r^2 \leq n \} \end{aligned}$$

2. Démontrer que, si ce programme termine, il calcule la racine carrée entière par défaut (preuve partielle).
3. Donner un variant (une fonction de terminaison) pour la boucle (12–15). Montrer que ce programme termine.
4. Donner un plan de test pour cette fonction.
5. Écrire une fonction `test_racine` conforme à votre plan de test.



## Écriture de code

**Exercice 4.** Si un train fait un trajet aller-retour entre 2 villes à la vitesse constante  $v_1$  pour l'aller et à la vitesse constante  $v_2$  au retour, la vitesse moyenne du trajet total n'est pas la moyenne arithmétique des deux vitesses, mais leur moyenne harmonique

$$\frac{2}{\frac{1}{v_1} + \frac{1}{v_2}}$$

Pour une série statistique discrète  $x = (x_1, \dots, x_p)$ , formée de  $p$  nombres  $> 0$ , la moyenne harmonique est définie par

$$G = \frac{p}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_p}} = \frac{p}{\sum_{i=1}^p \frac{1}{x_i}}.$$

Écrire une fonction `moyenne_harmonique` retournant la moyenne harmonique des éléments d'une liste non vide. On supposera la liste constituée uniquement de nombres strictement positifs.

**Exercice 5.** On considère un fichiers de notes d'étudiants `liste_etudiants.csv` dont voici un extrait:

```
11400130|Sarah|Croche|9.6,18.9
11402978|Gilbert|Lingot|14.2,14.2
11403526|Jean|Rage|10.3,9.7
11500571|Chantal|Gique|10.2,9.8
11502148|Jerry|Kahn|10.7,10.2
11503156|Aubin|Sahlor|15.1,13.9
11503188|Oscar|Ibou|11.2,12.8
11503442|Patrice|Taise|8.2,4.1
11505350|Alphonse|Urlotorouth|4.4,17.3
90000002|Terry|Dicule|0.0,6.2
90000003|Anne|Aconda|5.5,7.8
90000004|Théo|Jasmin|20.0,4.1
98001398|Ursule|Fureux|14.7,11.1
99990125|Harry|Zona|11.8,10.2
99990179|José|Patelefaire|7.5,6.3
```

Chaque ligne contient le numéro d'identification de l'étudiant (NIP), son prénom, son nom, sa note à l'épreuve de mathématique, sa note à l'épreuve d'informatique. Ces différents champs sont séparés par le caractère `|`, à l'exception des notes qui sont séparées par une virgule. Ce fichier est trié par NIP croissant.

1. Écrire un code Python ouvrant le fichier `liste_etudiants.csv` et qui définit quatre listes comme variables globales.

- `liste_NIP`, la liste des NIP des étudiants (nombres entiers)
- `liste_nom`, la liste des noms des étudiants sous la forme "Prénom Nom"
- `liste_note_math`, la liste des notes de mathématique (nombres flottants).
- `liste_note_info`, la liste des notes d'informatique (nombres flottants).

L'ordre des éléments de ces listes étant celui du fichier. On obtiendra donc les listes suivantes (seul les premiers termes sont représentés).

#### Python 3 Shell

```
>>> liste_NIP
[11400130, 11402978, 11403526, ...]
>>> liste_nom
['Sarah Croche', 'Gilbert Lingot', 'Jean Rage', ...]
>>> liste_note_math
[9.6, 14.2, 10.3, ...]
>>> liste_note_info
[18.9, 14.2, 9.7, ...]
```

2. Écrire une fonction `recherche_NIP` ayant pour argument une chaîne de caractère `nom` représentant le nom l'étudiant sous la forme "Prénom Nom". Cette fonction retourne

- le numéro d'identification étudiant si celui-ci apparaît sur la liste,
- l'entier 0 sinon.

#### Python 3 Shell

```
>>> recherche_NIP("Aubin Sahalor")
11503156
>>> recherche_NIP("Ray Défess")
0
```

3. Écrire une fonction `recherche_position` ayant pour argument un entier NIP, un numéro d'étudiant valide, et qui retourne la position de ce NIP dans la liste `liste_NIP`.

Puisque la liste `liste_NIP` est triée par ordre monotone croissant, *on utilisera une recherche dichotomique*.

#### Python 3 Shell

```
>>> recherche_position(11403526)
2
```

4. Expliciter un invariant permettant de prouver que votre fonction `recherche_position` est correcte (on ne demande pas les détails de la preuve).
5. Écrire une fonction `admis` ayant pour argument un flottant `barre`. Cette fonction retourne la liste des noms des étudiants dont la moyenne des notes de math et d'informatique dépasse la valeur `barre`.

#### Python 3 Shell

```
>>> admis(14)
['Sarah Croche', 'Gilbert Lingot', 'Aubin Sahalor']
>>> admis(18)
[]
>>> admis(10)
['Sarah Croche', 'Gilbert Lingot', 'Jerry Kahn',
'Aubin Sahalor', 'Oscar Ibou', 'Alphonse Urlotorouth',
'Théo Jasmin', 'Ursule Fureux', 'Harry Zona']
>>>
```

6. Quelle est la complexité en temps de la fonction `admis` ?

---

## DS 4 : Concours Blanc PTSI 1-2

---

### Exercice 1 Un système de vote : le vote par préférence

Dans cet exercice, les graphes ont un ensemble de sommets de la forme  $\{0, 1, \dots, n-1\}$  et sont orientés complets et pondérés par des entiers : pour tout couple  $(i, j)$  de sommets distincts, il existe un arc de  $i$  à  $j$  de poids  $m_{i,j} \in \mathbb{Z}$ . Le graphe est représenté par sa matrice d'adjacence  $M = (m_{i,j})_{0 \leq i, j \leq n}$  avec la convention que  $m_{i,i} = 0$  pour tout sommet  $i$  (il n'y a pas d'arc d'un sommet vers lui-même).

On considère une élection, à laquelle se présentent  $n$  candidats. Chaque électeur inscrit sur son bulletin l'ensemble des candidats (tous les candidats sont classés), par ordre de préférence. L'ensemble des bulletins est rassemblé dans une urne. Le nombre de votants est noté  $p$ , l'urne contient donc  $p$  bulletins.

Par exemple, s'il y a trois candidats et qu'un électeur préfère le candidat 2, puis le candidat 0 et enfin considère que le candidat 1 est le moins souhaitable, son bulletin de vote sera  $[2;0;1]$ .

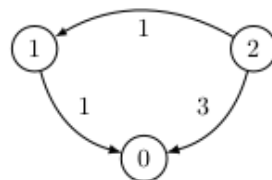
Une fois le vote effectué dans l'urne  $u$ , on compare les résultats de deux candidats particuliers  $i$  et  $j$  en comptant le nombre de bulletins qui classent le candidat  $i$  avant le candidat  $j$ . On exprime le résultat de la comparaison entre les candidats  $i$  et  $j$  en calculant la différence entre le nombre de bulletins de vote qui placent  $i$  avant  $j$  et le nombre de ceux qui placent  $j$  devant  $i$ .

Exemple 1 : On considère une élection avec trois candidats et trois votants :  $n = 3$  et  $p = 3$ . L'urne  $u$  est composée des trois bulletins  $[2;0;1]$ ,  $[2;1;0]$  et  $[1;2;0]$ . L'urne  $u$  est donc la liste  $u = [[2;0;1], [2;1;0], [1;2;0]]$ .

La comparaison entre le candidat numéro 0 et le candidat numéro 1 donne  $-1$  car le candidat est placé une fois avant le candidat 1 et deux fois après.

On peut alors synthétiser le contenu d'une urne  $u$  en construisant le graphe de préférence des votants : ses sommets correspondent aux candidats et l'arc du sommet  $i$  vers le sommet  $j$  est pondéré par la comparaison entre le candidat  $i$  et le candidat  $j$ .

La figure suivante donne le graphe de préférence obtenu à partir de l'exemple 1 ainsi que sa matrice d'adjacence  $M$ . Afin d'alléger le schéma, seuls les arcs avec un poids strictement positifs sont représentés.



$$M = \begin{pmatrix} 0 & -1 & -3 \\ 1 & 0 & -1 \\ 3 & 1 & 0 \end{pmatrix}$$

1. Ecrire une fonction `compte` prenant en argument un bulletin  $b$  et deux candidats  $i$  et  $j$  et retournant `True` si le candidat  $i$  est placé avant le candidat  $j$  dans le bulletin  $b$ , `False` sinon.

- 
2. Ecrire une fonction `duel` prenant en argument une urne  $u$  et deux candidats  $i$  et  $j$  et retournant la comparaison entre les candidats  $i$  et  $j$  pour tous les bulletins de l'urne  $u$ . Dans l'exemple 1, on avait par exemple `duel(u, 0, 1) = -1`.
  3. Exemple 2 : On considère une élection avec trois candidats et quatre votants :  $n = 3$  et  $p = 4$ . A l'issue du vote, le contenu de l'urne  $u$  est  $[[0;1;2], [1;2;0], [0;2;1], [0;2;1]]$ . Tracer le graphe de préférence de l'urne (en ne dessinant que les arcs ayant un poids strictement positif) et donner sa matrice d'adjacence.
  4. Expliquer pourquoi la matrice d'adjacence d'un graphe de préférence est antisymétrique et pourquoi tous ses coefficients non-diagonaux ont la même parité.

Etant donné une urne  $u$ , on note  $\text{Mat}(u)$  la matrice du graphe de préférence associée à cette urne.

5. Ecrire une fonction `depouillement` prenant en argument le nombre de candidats  $n$  et une urne  $u$  et retournant la matrice  $\text{Mat}(u)$ .
6. Le but de cette sous-partie est de démontrer le théorème de McGarvey : "Pour toute matrice antisymétrique à coefficients entiers pairs  $M$ , il existe une urne  $u$  telle que  $M = \text{Mat}(u)$ ".  
On rappelle qu'une matrice  $M$  est dite antisymétrique si elle vérifie  $M^T = -M$  où  $M^T$  est sa transposée et on note  $\mathcal{A}_n$  l'ensemble des matrices réelles antisymétriques de taille  $n$ .

(a) Montrer que  $\mathcal{A}_n$  est un espace vectoriel.

Soient  $i, j$  et  $n$  trois entiers naturels tels que  $i < n, j < n$  et  $i \neq j$ . On note  $E_{i,j,n}$  la matrice carrée de taille  $n$  dont tous les coefficients sont nuls sauf les coefficients d'indices  $(i, j)$  et  $(j, i)$  qui valent respectivement 2 et  $-2$ .

- (b) Montrer que la famille  $(E_{i,j,n})_{1 \leq i < j \leq n}$  est une base de  $\mathcal{A}_n$ . En déduire sa dimension.
  - (c) Soient  $n$  un nombre de candidats et  $i$  et  $j$  deux entiers naturels strictement inférieurs à  $n$ . Montrer qu'il existe une urne  $u_{i,j,n}$  contenant deux votes telles que  $\text{Mat}(u_{i,j,n}) = E_{i,j,n}$ .
  - (d) On considère deux urnes  $u_1$  et  $u_2$ . Exprimez  $M_3 = \text{Mat}(u_1 \cup u_2)$  en fonction de  $M_1 = \text{Mat}(u_1)$  et  $M_2 = \text{Mat}(u_2)$ .
  - (e) Démontrer le théorème de McGarvey.
  - (f) Ecrire une fonction `mcgarvey` prenant en argument une matrice antisymétrique  $M$  de coefficients tous pairs et retournant une urne  $u$  telle que  $M = \text{Mat}(u)$ .
  - (g) Estimer la complexité de la fonction `mcgarvey` en fonction de  $n$ , la taille de la matrice (c'est-à-dire le nombre de candidats), et de  $q$ , le maximum des coefficients de la matrice.
7. On appelle vainqueur de Condorcet tout sommet tel que, dans le graphe des préférences, les arcs sortant de ce sommet ont tous un poids positif ou nul. Ainsi, dans l'exemple 1, le candidat 2 est un vainqueur de Condorcet.
    - (a) Expliquer pourquoi un vainqueur de Condorcet peut être qualifié de "vainqueur" de l'élection.
    - (b) Ecrire une fonction `condorcet` prenant en argument la matrice d'adjacence d'un graphe et retournant la liste des vainqueurs de Condorcet.

- 
- (c) En se plaçant dans le cas  $n = 3$  et  $p = 3$ , construire une urne pour laquelle il n'existe pas de vainqueur de Condorcet. Tracer le graphe de préférence correspondant.

**Exercice 2** Dans cet exercice, on manipule des suites (finies) d'entiers sous forme de listes d'entiers. Ainsi la suite  $(0, 1, 2, 3)$  sera représentée par la liste  $[0, 1, 2, 3]$ . La liste est croissante (respectivement décroissante, monotone) si la suite est croissante (respectivement décroissante, monotone).

1. Ecrire une fonction **estCroissante** d'argument une liste d'entiers et qui renvoie un booléen indiquant si cette liste est croissante ou pas. Cette fonction devra être de complexité linéaire dans le pire des cas.
2. Ecrire de même des fonctions **estDecroissante** et **estMonotone** qui testent si une liste d'entiers est respectivement décroissante, monotone.

Soit la liste  $L = [u_0, u_1, \dots, u_{n-1}]$  de longueur  $n$ . On appelle tranche de  $L$  une liste de la forme  $[u_i, u_{i+1}, \dots, u_j]$  où  $0 \leq i \leq j < n$ . On cherche une tranche de  $L$  croissante et de longueur maximale. Par exemple, une tranche croissante de longueur maximale de  $[0, 1, 0, 1, 2, 3, 4, 0, 1, 2]$  est  $[0, 1, 2, 3, 4]$ , correspondant aux indices  $i = 2$  à  $j = 6$ .

3. Ecrire une fonction **LC** de deux arguments, une liste  $L$  et un entier  $p$ , qui renvoie l'entier  $d$  tel que : la liste  $[u_p, \dots, u_{p+d}]$  est croissante et soit  $p + d = n - 1$ , soit  $u_{p+d} > u_{p+d+1}$ .
4. Ecrire une fonction **maxCroissante** d'argument une liste  $L$  qui renvoie la plus longue tranche croissante de  $L$ . S'il n'y a pas unicité, on renvoie la première trouvée.