Lycée Faidherbe, 2019-2020

Devoirs d'informatique

MPSI et PCSI

Version du 20 janvier 2020

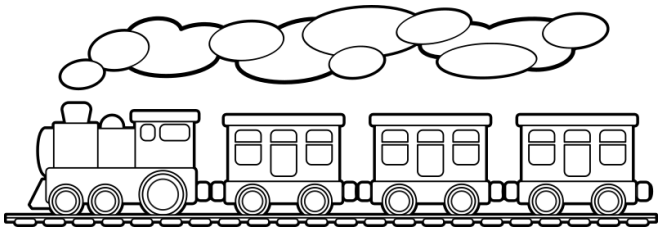
Table des matières

Devoir I

Trains (MPSI)

Recommandations

- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.
- Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.



1 Fréquentation d'une ligne de trains

Pour étudier la structure des trains et le nombre de départs qu'il peut être utile de prévoir la SNCF dispose du nombre de passagers ayant réservé une place dans une ligne de TGV (par exemple Lille-Paris) pour chaque journée.
On a donc des listes de nombres ; les fonctions seront accompagnées par leurs valeurs pour la suite non réaliste :

```
L0 = [34, 32, 27, 15, 21, 23, 29, 40, 35, 31, 12, 26, 25, 33]
```

- La longueur de la liste est le nombre de jours consécutifs étudiés.
- `L[0]` est le nombre de réservations pour le premier jour, `L[1]` pour le deuxième, ...
- On supposera que la longueur de la liste est toujours un multiple de 7.
- On supposera de plus que le premier jour de l'étude sera toujours un dimanche

Le but de cette partie est d'écrire des fonctions qui permettent d'extraire des informations depuis ces listes.

1.1 Fonctions globales

On définit

```
def xxx(liste):
    n = len(liste)
    a = liste[0]
    for i in range(1, n):
        if liste[i] > a:
            a = liste[i]
    return a
```

Exercice I.1 — Fonction inconnue
Que renvoie la fonction `xxx` *appliquée à une liste non vide ?*

Exercice I.2 — Minimum
Écrire une fonction `minimum(liste)` *qui renvoie le minimum des valeurs d'une liste.*
`minimum(L0)` renvoie `12`

Exercice I.3 — Moyenne
Écrire une fonction `moyenne(liste)` *qui renvoie la moyenne des valeurs d'une liste*

`moyenne(L0)` renvoie `27.357142857142858`

La **variance** d'une d'une suite de valeurs $(a_k)_{0\leqslant k<n}$ est la moyenne des carrées des écarts à la moyenne : $V = \dfrac{1}{n}\displaystyle\sum_{k=0}^{n-1}(a_k - m)^2$ avec $m = \dfrac{1}{n}\displaystyle\sum_{k=0}^{n-1} a_k$. L'**écart-type** est la racine de la variance.
Rappel : la racine de `a` peut se calculer, dans python, par `a**0.5`

Exercice I.4 — Écart-type
Écrire une fonction `ecart_type(liste)` *qui renvoie l'écart-type des valeurs d'une liste*

`ecart_type(L0)` renvoie `7.49863933235455`

Ces données donnent une idée de la dispersion de fréquentation.
On peut souhaiter aussi déterminer les jours exceptionnels où, par exemple, la fréquentation est supérieure (ou inférieure) à un seuil que l'on fixe. Par exemple le seuil peut être la capacité maximale des train ou $m + 2e$, si m est la moyenne et e l'écart-type.

Exercice I.5 — Nombre de dépassements supérieurs
Écrire une fonction `nb_plus(liste, a)` *qui renvoie le nombre d'indices* `i` *tels que* `liste[i] > a`.
`nb_plus(L0, 34)` renvoie `2`

Exercice I.6 — Dépassements inférieurs
Écrire une fonction `ind_moins(liste, a)` *qui renvoie la liste des indices* `i` *tels que* `liste[i] < a`.
`ind_moins(L0, 18)` renvoie `[3, 10]`

1.2 Découpage hebdomadaire

En fait la fréquentation varie beaucoup selon les jours de la semaine.
On va donc étudier la liste des réservations en tenant compte du jour ou, à l'inverse, en regrouppant par semaine pour gommer les disparités journalières.

Comme l'indice 0 correspond à un dimanche, le premier mardi (par exemple) correspond à l'indice 2 et tous les mardi sont associés aux indices de la forme $2 + 7k$; la liste des réservations ayant eu lieu un mardi est ainsi donnée par `liste[2 : n : 7]` si n est la longueur de la liste.

Exercice I.7 — Moyenne par jour
Écrire une fonction `moyenne_jour` *telle que, si* `liste` *est une liste des réservations pour chaque jour, le premier jour étant un dimanche,* `J = moyenne_jour(liste)` *est une liste de longueur 7 telle que* `J[0]` *est la moyenne des réservations des dimanches,* `J[1]` *la moyenne des lundi et ainsi de suite jusqu'au samedi pour* `J[6]`.

`moyenne_jour(L0)` renvoie `[37.0, 33.5, 29.0, 13.5, 23.5, 24.0, 31.0]`

On suppose qu'on a défini la liste des jours :

```
jours = ["dimanche",    "lundi",    "mardi", "mercredi",
                        "jeudi", "vendredi",   "samedi"]
```

Cette liste est considérée comme une constante, on pourra l'utiliser dans une fonction sans la définir.

Exercice I.8 — Fréquentation maximale
Écrire une fonction `jour_max(liste)` *qui renvoie le nom du premier jour pour lequel le nombre de réservations moyen est maximal.*

`jour_max(L0)` renvoie `"dimanche"`

On passe maintenant aux totaux par semaine.

Exercice I.9 — Total hebdomadaire
Écrire une fonction `totaux_semaine` *qui reçoit une liste comme paramètre et qui renvoie la liste des totaux de réservation pour chaque semaine : si* `sem = totaux_semaine(liste)` *alors* `sem[0]` *doit être le total des valeurs de* `liste[i]` *pour i variant de 0 à 6,* `sem[1]` *doit être le total des valeurs de* `liste[i]` *pour i variant de 7 à 13 et ainsi de suite.*
On rappelle que la longueur de la liste doit être un multiple de 7.

`totaux_semaine(L0)` renvoie `[181, 202]`

2 Retards

La SNCF note aussi les heures d'arrivée sous la forme des écarts par rapport à l'heure prévue.
On dispose donc d'une liste d'entiers qui donne cet écart en minutes, positif ou négatif, pour chaque jour : par exemple `E0 = [2, 15, 1, -2, 6, 4, 3, 0, 7, 2, -2, 1, 1]`.
Un écart positif signifie que le train est arrivé en retard,
un écart négatif signifie que le train est arrivé en avance.

On souhaite déterminer le nombre de trains en retard, ce sont ceux pour lesquels l'écart est supérieur à un seuil : le retard admissible (par exemple 1 minute).

Exercice I.10 — Nombre de retards
Écrire une fonction `nb_retards(liste, seuil)` *qui renvoie le nombre de valeurs de la liste strictement supérieures au seuil.*

`nb_retards(E0, 2)` renvoie `5`

Un retard est un écart positif, les écarts négatifs "ne comptent pas".

Exercice I.11 — Retard moyen
Écrire une fonction `moy_retards(liste)` *qui renvoie la moyenne des retards, c'est-à-dire, la somme des termes positifs divisée par la longueur de la liste.*

`moy_retards(E0)` renvoie `3.230769230769231`

On s'intéresse aux durées sans retard (au-delà d'un seuil).

Exercice I.12 — Pointage des retards
Écrire une fonction `pointage(liste, seuil)` *qui renvoie la liste des indices* `i` *tels que* `liste[i]` *est strictement supérieur au seuil.*

`pointage(E0,2)` renvoie `[1, 4, 5, 6, 8]`

Exercice I.13 — Pointage des retards avec bornes
Modifier la fonction `pointage` *en une fonction* `pointage1(liste, seuil)` *afin qu'elle renvoie la liste des valeurs définies ci-dessus précédées de* -1 *et suivie de* n *où est la longueur de la liste.*

`pointage1(E0,2)` renvoie `[-1, 1, 4, 5, 6, 8, 13]`

Exercice I.14 — Plage maximale sans retard
Écrire une fonction `plage_max(liste, seuil)` *qui renvoie le plus grand entier* k *tel qu'il existe des indices* a *et* b *avec* $b = a+k-1$ *pour lesquels on a* `liste[i] <= seuil` *pour tout* $i \in \{a, a+1, \dots, b\}$.
Si toutes les valeurs sont supérieures au seuil, la fonction renvoie 0.

`plage_max1(E0,2)` renvoie `4`
On pourra utiliser les fonctions ci-dessus mais il existe plusieurs méthodes

Exercice I.15 — Plage maximale sans retard
Modifier la fonction `plage_max` *en une fonction* `bornes_plage_max(liste, seuil)` *qui renvoie les indices* a *et* b *définis ci-dessus.*
Si toutes les valeurs sont supérieures au seuil, la fonction renvoie 0, -1.

`bornes_plage_max1(E0,2)` renvoie `9, 12`

3 Taille de la gare

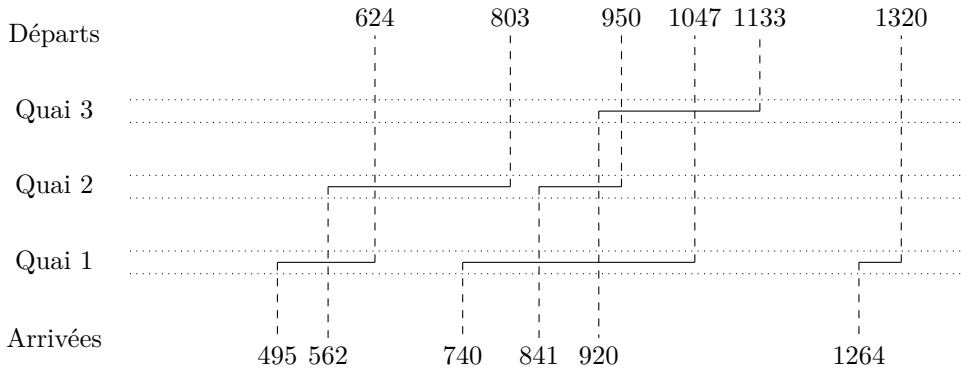
Dans cette question, on s'intéresse au nombre de quais de gare qu'il est nécessaire de disposer pour pouvoir laisser les trains en attente sans les déplacer.

- Pour simplifier les horaires sont donnés en minutes à partir de minuit : 8h37 sera indiqué 517 et 19h51 sera noté 1191.
- On dispose des horaires d'arrivée et des horaires de départ sous la forme de deux listes croissantes.
- Les trains sont interchangeables : on peut faire partir à l'heure indiquée n'importe quel train parmi ceux qui sont à quai.
- On suppose qu'il y a toujours assez de trains arrivés pour les départs prévus.
- On suppose que les horaires d'arrivées et de départs sont tous distincts.
- Il y a autant de trains au départ qu'à l'arrivée.

Par exemple si la liste des arrivées et la liste des départ sont

```
arrivees0 = [ 495,  562,  740,  841,  920, 1264]
departs0  = [ 624,  803,  950, 1047, 1133, 1320]
```

On doit avoir au moins 3 quais disponibles.



Exercice I.16 — Nombre de quais
Écrire une fonction `nb_quais(arrivees, departs)` *qui renvoie le nombre de quais nécessaires au trafic déterminé par les deux listes d'arrivées et de départs.*

`nb_quais(arrivees0, departs0)` renvoie `3`

4 Solutions

Solution de l'exercice I.1 - La fonction renvoie la maximum de la liste.

Solution de l'exercice I.2 -

```
def minimum(liste):
    n = len(liste)
    a = liste[0]
    for i in range(1, n):
        if liste[i] < a:
            a = liste[i]
    return a
```

Solution de l'exercice I.3 -

```
def moyenne(liste):
    n = len(liste)
    s = 0
    for i in range(n):
        s = s + liste[i]
    return s/n
```

Solution de l'exercice I.4 -

```
def ecart_type(liste):
    n = len(liste)
    m = moyenne(liste)
    s = 0
    for i in range(n):
        s = s + (liste[i] - m)**2
    return (s/n)**0.5
```

Solution de l'exercice I.5 -

```
def nb_plus(liste, a):
    n = len(liste)
    nb = 0
    for i in range(n):
        if liste[i] > a:
            nb = nb + 1
    return nb
```

Solution de l'exercice I.6 -

```
def ind_moins(liste, a):
    n = len(liste)
    ind = []
    for i in range(n):
        if liste[i] < a:
            ind.append(i)
    return ind
```

Solution de l'exercice I.7 -

```
def moyenne_jour(liste):
    J = [0]*7
    for i in range(7):
        J[i] = moyenne(liste[j : : 7])
    return J
```

Solution de l'exercice I.8 -

```
def jour_max(liste):
    J = moyenne_jour(liste)
    j_max = 0
    for i in range(1, 7):
        if J[i] > J[j_max]:
            j_max = j
    return jours[j_max]
```

Solution de l'exercice I.9 -

```
def totaux_semaine(liste):
    n = len(liste)
    p = n//7
    sem = [0]*p
    for i in range(n):
        k = i//7
        sem[k] = sem[k] + liste[i]
    return sem
```

Solution de l'exercice I.10 -

```
def nb_retards(liste, seuil):
    nb = 0
    for x in liste:
        if x > seuil:
            nb  = nb + 1
    return nb
```

Solution de l'exercice I.11 -

```
def moy_retards(liste):
    n = len(liste)
    s = 0
    for x in liste:
        if x > 0:
            s = s + x
    return s/n
```

Solution de l'exercice I.12 -

```
def pointage(liste, seuil):
    n = len(liste)
    p = []
    for i in range(n):
        if liste[i] > seuil:
            p.append(i)
    return p
```

Solution de l'exercice I.13 -

```
def pointage1(liste, seuil):
    n = len(liste)
    p = [-1]
    for i in range(n):
        if liste[i] > seuil:
            p.append(i)
    p.append(n)
    return p
```

Solution de l'exercice I.14 -

```
def plage_max(liste, seuil):
    k = 0
    p = pointage1(liste, seuil)
    n = len(p)
    for i in range(n-1):
        if p[i+1] - p[i] -1 > k:
            k = p[i+1] - p[i] - 1
    return k
```

Une méthode plus directe.

```
def plage_max(liste, seuil):
    k_max = 0
    k = 0
    n = len(liste)
    for x in liste:
        if x > seuil:
            k = 0
        else:
            k = k + 1
            if k > k_max:
                k_max = k
    return k_max
```

Solution de l'exercice I.15 -

```
def bornes_plage_max(liste, seuil):
    a = 0
    b = -1
    p = pointage1(liste, seuil)
    n = len(p)
    for i in range(n-1):
        if p[i+1] - p[i] > b - a + 2:
            a = p[i] + 1
            b = p[i+1] - 1
    return a, b
```

```
def bornes_plage_max(liste, seuil):
    a_max = 0
    b_max = -1
    a = 0
    b = 0
    n = len(liste)
    for i in range(n):
        if liste[i] > seuil:
            a = i + 1
        else:
            b = i
            if b - a > b_max - a_max:
                a_max = a
                b_max = b
    return a_max, b_max
```

Solution de l'exercice I.16 -

```
def nb_quais(arrivees, departs):
    n = len(arrivees)
    ia = 0
    id = 0
    nb = 0
    nb_max = 0
    while ia < n:
        if arrivees[ia] < departs[ib]:
            nb = nb + 1
            ia = ia + 1
            if nb > nb_max:
                nb_max = nb
        else:
            nb = nb - 1
            ib = ib + 1
    return nb_max
```

Devoir II

Classification périodique (PCSI)

Recommandations

- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.
- Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.

La classification périodique est un tableau regroupant les éléments chimiques classés par famille (voir figure II.1). Pour chaque élément chimique, on associe un numéro atomique Z (qui est le nombre de proton et d'électron) et un nom. Ainsi, on voit que l'atome de numéro atomique `Z=20` est le calcium `Ca`.

On se limitera dans toute la suite aux 54 premiers éléments de la classsification ($1 \leqslant Z \leqslant 54$).

n																	
1	1 H																2 He
2	3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
3	11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 C	18 Ar
4	19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
5	37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe

Figure II.1 – Classification périodique, 54 premiers éléments

1 Introduction

On se donne la liste `L1` donnée ci-dessous :

```
L1 = ["H",  "He",  "Li",  "Be",  "B",   "C",   "N",   "O",   "F",
      "Ne",  "Na",  "Mg",  "Al",  "Si",  "P",   "S",   "C",   "Ar",
      "K",   "Ca",  "Sc",  "Ti",  "V",   "Cr",  "Mn",  "Fe",  "Co",
      "Ni",  "Cu",  "Zn",  "Ga",  "Ge",  "As",  "Se",  "Br",  "Kr",
      "Rb",  "Sr",  "Y",   "Zr",  "Nb",  "Mo",  "Tc",  "Ru",  "Rh",
      "Pd",  "Ag",  "Cd",  "In",  "Sn",  "Sb",  "Te",  "I",   "Xe"]
```

Exercice II.1
Soit `Z` *un entier compris entre 1 et 54. Que renvoie* `L1[Z-1]` *?*

Les éléments chimique sont rangés dans le tableau périodique par l'intermédiaire du nombre quantique principal n (avec $n \geqslant 1$). On constate que :

- Pour $Z = 1$ ou 2, $n = 1$;
- Pour $3 \leqslant Z \leqslant 10$, $n = 2$;
- Pour $10 \leqslant Z \leqslant 18$, $n = 3$;
- Pour $19 \leqslant Z \leqslant 36$, $n = 4$;
- Pour $37 \leqslant Z \leqslant 54$, $n = 5$.

Exercice II.2
Écrire une fonction `periode(Z)` *qui prend en argument un entier* `Z` *compris entre 1 et 54 et qui renvoie l'entier* `n` *correspondant au nombre quantique principal.*

2 Orbitale atomique

En mécanique quantique, on constate que les électrons d'un atome occupent des orbitales atomiques. Une orbitale est caractérisée par deux nombres : n (le nombre quantique principal) et ℓ le nombre secondaire (avec $0 \leqslant \ell \leqslant n$). De plus, on associe une lettre aux nombres secondaires

- à $\ell = 0$, on associe le nom `"s"` ;
- à $\ell = 1$, on associe le nom `"p"` ;
- à $\ell = 2$, on associe le nom `"d"` ;
- à $\ell = 3$, on associe le nom `"f"` ;

et on note sous la forme d'une chaîne de caractère à deux signes une orbitale : par exemple, l'orbitale `"2p"` correspond aux nombres $n = 2$ et $\ell = 1$. La fonction `nom_orbitale` transforme un couple d'entiers représentant une orbitale en la chaîne associée.

```
def nom_orbitale(n,l):
    L2 = ["s","p","d","f"]
    return str(n) + L2[l]
```

Exercice II.3
Que renvoie `nom_orbitale(4, 0)` *?*

Exercice II.4
Écrire une fonction `nombres_quantiques` *qui réalise l'opération inverse et renvoie la liste composée de n et ℓ à partir d'une chaîne de 2 caractères.*
Ainsi, `nombres_quantiques("3s")` *doit renvoyer* `[3,0]`.

Une orbitale atomique de nombres quantiques n et ℓ peut contenir au maximum $4\ell + 2$ électrons.

Exercice II.5
Écrire une fonction `nombre_max(l)` *qui renvoie le nombre maximum d'électrons que peut contenir une orbitale atomique de nombre quantique secondaire ℓ.*

3 Structure électronique

On cherche dans la suite à construire la structure électronique d'un atome dans son état fondamental. Par exemple, pour l'atome du l'antimoine Sb ($Z = 51$), elle est notée :

$$1s^2\ 2s^2\ 2p^6\ 3s^2\ 3p^6\ 4s^2\ 3d^{10}\ 4p^6\ 5s^2\ 4d^{10}\ 5p^3$$

Cela veut dire que l'orbitale 1s contient 2 électrons, la 2s aussi, la 2p contient 6 électrons et ainsi de suite jusqu'à la 5p qui contient 3 électrons. On se donne dans la suite la liste :

```
L3 = ["1s","2s","2p","3s","3p","4s","3d","4p","5s","4d","5p"]
```

qui correspond aux orbitales utiles dans la suite. Celle-ci est ordonnées dans le sens de remplissage. Autrement dit, on va attribuer prioritairement les électrons à l'orbitale `"1s"` qui peut en contenir au maximum 2, puis à l'orbitale `"2s"` et ainsi, de suite.

Exercice II.6
Écrire une fonction `configuration(Z)` *qui renvoie une liste de même taille que* `L3` *correspondant au remplissage des orbitales. Ainsi,* `configuration(51)` *doit renvoyer*

`[2, 2, 6, 2, 6, 2, 10, 6, 2, 10, 3]`.

On pourra utiliser les fonctions des questions 4 et 5, même si on n'y a pas répondu.

On aimerait, à présent que le programme écrive une phrase du type :

```
"Atome Sb (Z=51) : 1s2-2s2-2p6-3s2-3p6-4s2-3d10-4p6-5s2-4d10-5p3"
```

Exercice II.7
Écrire une fonction `chaine_conf(Z)` *qui renvoie une chaîne de caractères comme ci-dessus.*

4 Raffinement

Le programme précédent ne nous convient pas totalement. En effet, si l'on prend l'exemple du lithium ($Z = 3$), celui-ci nous renvoie une configuration $1s^2\ 2s^1\ 2p^0\ 3s^0\ 3p^0\ 4s^0\ 3d^0\ 4p^0\ 5s^0\ 4d^0\ 5p^0$ alors qu'on préférerait qu'il nous renvoie simplement $1s^2\ 2s^1$!

Exercice II.8
Écrire une fonction `configuration_simple(Z)` *qui réalise cette instruction en renvoyant une liste. Ainsi,* `configuration_simple(3)` *doit renvoyer* `[2,1]`.

Exercice II.9
Écrire une fonction `chaine_conf_simple(Z)` *qui est une version modifiée de* `chaine_conf(Z)` *tenant compte de la question précédente. Ainsi,* `chaine_conf_simple(3)` *doit renvoyer la chaîne de caractères*

`"Atome Li (Z=3) : 1s2-2s1"`

Exercice II.10
Écrire une fonction `ensemble_configuration()` *qui ne prend aucun argument, qui ne renvoie rien mais qui imprime la configuration des 54 premiers éléments, c'est-à-dire :*

```
Atome H (Z=1) : 1s1
Atome He (Z=2) : 1s2
Atome Li (Z=3) : 1s2-2s1
...
```

5 Solutions

Solution de l'exercice II.1 - La fonction renvoie la valeur stockée à la case d'indice $Z-1$, c'est-à-dire le nom de l'atome de numéro atomique Z.

Solution de l'exercice II.2 -

```
def periode(Z):
    if Z<=2:
        return 1
    elif Z<=10:
        return 2
    elif Z<=18:
        return 3
    elif Z<=36:
        return 4
    else:
        return 5
```

Solution de l'exercice II.3 - On trouve la chaîne de caractères `"4s"`.

Solution de l'exercice II.4 -

```
def nombres_quantiques(x):
    n = int(x[0])
    for i in range(len(L2)):
        if L2[i] == x[1]:
            l = i
    return [n,l]
```

Solution de l'exercice II.5 -

```
def nombre_max(l):
    return 4*l + 2
```

Solution de l'exercice II.6 -

```
def configuration(Z):
    p = len(L3)
    config = [0]*p
    for i in range(p):
        n, l = nombres_quantiques(L3[i])
        possible = nombre_max(l)
        couche  = min(Z, possible)
        config[i] = couche
        Z = Z - couche
    return config
```

Solution de l'exercice II.7 -

```
def chaine_conf(Z):
    config = configuration(Z)
    n = len(config)
    nom = L1[Z-1]
    phrase = "Atome {} (Z = {} : ".format(nom, Z)
    for i in range(n - 1):
        phrase = phrase + L3[i] + str(config[i]) + "-"
    return phrase  + L3[n-1] + str(config[n-1])
```

Solution de l'exercice II.8 - On remplace la boucle `for` par une boucle `while`.

```
def configuration_simple(Z):
    p = len(L3)
    config = []
    i = 0
    while i < p and Z > 0:
        n, l = nombres_quantiques(L3[i])
        possible = nombre_max(l)
        couche  = min(Z, possible)
        config.append(couche)
        Z = Z - couche
        i = i + 1
    return config
```

Solution de l'exercice II.9 - C'est la même fonction qu'à l'exercice 7

```
def chaine_conf(Z):
    config = configuration_simple(Z)
    n = len(config)
    nom = L1[Z-1]
    phrase = "Atome {} (Z = {} : ".format(nom, Z)
    for i in range(n - 1):
        phrase = phrase + L3[i] + str(config[i]) + "-"
    return phrase  + L3[n-1] + str(config[n-1])
```

Solution de l'exercice II.10 -

```
def ensemble_configuration():
    for i in range(1,55):
        print(chaine_conf_simple(i))
```

Devoir III

Exercices (PCSI)

- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.
- Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.

1 Exercices tirés des TP

Exercice III.1 — Une suite
Écrire une fonction `heron(n,u0)` qui calcule (et renvoie) le terme u_n de la suite (u_p) définie par $u_0 =$ `u0` et $u_{p+1} = \frac{1}{2}\left(u_p + \frac{2}{u_p}\right)$ (suite de Héron).

Exercice III.2 — Suite de Fibonacci : 1
On considère la suite (de Fibonacci) définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour $n \geqslant 0$.
Écrire un programme `fibo(n)` qui renvoie F_n.
On pourra maintenir deux variables qui correspondent à 2 valeurs consécutives de la suite.

Exercice III.3 — Second degré
Écrire une fonction `racines(a, b, c)` qui, étant donnée une équation du second degré $ax^2 + bx + c = 0$ identifiée par ses 3 coefficients réels, renvoie la liste de ses racines réelles.
La liste retournée doit donc avoir 2, 1 ou 0 éléments selon que le polynôme admet 2 racines réelles distinctes, une racine double ou n'admet pas de racine réelle.
On supposera, sans avoir besoin de le vérifier, que a est non nul.

Exercice III.4 — Produit
Écrire une fonction `produit(liste)` qui calcule le produit des termes d'une liste.

Exercice III.5 — Degré
Écrire une fonction `degre(P)` *qui renvoie le degré d'un polynôme donné en paramètre sous la forme d'une liste : le polynôme* $P = \sum_{k=0}^{n} a_k X^k$ *où les* a_k *sont des réels est représenté par la liste* $[a_0, a_1, \cdots a_n]$ *de ses coefficients.*
Par convention, doit renvoyer -1 *si le polynôme est nul.*

2 Développements

Exercice III.6 — Suite de Fibonacci : 2
On considère la suite (de Fibonacci) définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour $n \geqslant 0$.
Compléter le programme suivant afin qu'il renvoie la liste des nombres de Fibonacci de F_0 à F_n.

```
def liste_fibo(n):
    """Entrée : un entier positif n
       Sortie : la liste des Fk pour k de 0 à n"""
    F = [0]*----- # ce n'est pas F = [0]*n
    F[0] = -----
    F[1] = -----
    for i in range(2, -----):
        F[i] = -----
    return F
```

Exercice III.7 — Test
On veut vérifier que tous les termes d'une liste appartiennent à un intervalle $[a; b]$.
Écrire une fonction `encadrement(liste, a, b)` qui renvoie `True` si tous les termes de la liste vérifient `a <= liste[i] <= b` et `False` si au moins un terme ne vérifie pas cette propriété.

Exercice III.8 — Coefficients binomiaux
On veut calculer la liste des coefficients binomiaux $\binom{n}{p}$ pour n fixé et p variant de 0 à n.
Déterminer $\dfrac{\binom{n}{p+1}}{\binom{n}{p}}$ pour $0 \leqslant p < n$.
En déduire une fonction `liste_binom(n)` qui calcule la liste souhaitée **sans** calculer les factorielles.
On prendra soin d'utiliser la division entière : `//`.

Exercice III.9 — Somme de polynômes
On veut calculer la sommes de deux polynômes définis par leur liste de coefficients comme dans l'exercice III.5. On notera que ces listes peuvent ne pas avoir la même longueur.
Écrire une fonction `somme_poly(P, Q)` qui renvoie la liste représentant la somme des polynômes.

3 Quais de gare

On s'intéresse au nombre de quais d'une gare qui sont nécessaires pour pouvoir recevoir les trains en attente entre leurs arrivées et leur départs.

- Pour simplifier les horaires sont donnés en minutes à partir de minuit : 8h37 sera indiqué $8 \times 60 + 37 = 517$ et 19h51 sera noté $19 \times 60 + 51 = 1191$.
- Les horaires d'arrivée et des horaires de départ sont donnés par deux listes croissantes.
- Les trains sont interchangeables : on peut faire partir à l'heure indiquée n'importe quel train parmi ceux qui sont à quai.
- On suppose qu'il y a toujours assez de trains arrivés pour les départs prévus.
- On suppose que les horaires d'arrivées et de départs sont tous distincts.

Par exemple si la liste des arrivées et la liste des départ sont

```
arrivees0 = [ 495,  562,  740,  841,  920, 1264]
departs0  = [ 624,  803,  950, 1047, 1133, 1320]
```

3 quais sont nécessaires car, entre les temps 920 et 950, il y a 3 trains arrivés et non partis.

Exercice III.10 — Nombre de quais
Écrire une fonction `nb_quais(arrivees, departs)` *qui renvoie le nombre de quais nécessaires au trafic déterminé par les deux listes d'arrivées et de départs.*

`nb_quais(arrivees0, departs0)` renvoie 3

4 Solutions

Solution de l'exercice III.1 -

```
def heron(n,u0):
      """Entrées : un entier positif n et un réel u0
         Sortie  : le n-ième terme de la suite de Héron"""
    u = u0
    for i in range(n):
        u = (u + 2/u)/2
    return u
```

Solution de l'exercice III.2 -

```
def fibo(n):
    """Entrée : un entier positif n
       Sortie : le n-ième nombre de Fibonacci"""
    F = 0
    F_suivant = 1
    for i in range(n): # On calcule les couples (F1, F2), (F2,
        F3), ..., (Fn,Fn+1)
        F_vieux = F
        F = F_suivant
        F_suivant = F + F_vieux
    return F
```

Solution de l'exercice III.3 -

```
def racinesReelles(a, b, c):
    """Entrees : 3 flottants representant
                 le polynome aX**2 + bX + c
       Sortie : la liste des racines reelles"""
    delta = b**2 - 4*a*c
    if delta > 0:
        return [(-b + delta)/2*a, (-b - delta)/(2*a)]
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return []
```

Solution de l'exercice III.4 - Ne pas oublier d'initialiser le produit à 1 et non à 0.

```
def produit(liste):
    """Entrée : une liste
       Sortie : le produit des termes de la liste"""
    prod = 1          # On initialise le produit
    for x in liste:   # Chaque élément de la liste
        prod = prod*x # est multiplié
    return somme
```

Solution de l'exercice III.5 -

```
def degre(P):
    """Entrée : une liste qui représente un polynome
       Sortie : le degré du polynôme,
                -1 pour le polynome nul"""
    deg = -1
    for k in range(len(P)):
        if P[k] != 0:
            deg = k
    return deg
```

Solution de l'exercice III.6 -

```
def liste_fibo(n):
    """Entrée : un entier positif n
       Sortie : les n premiers nombres de Fibonacci"""
    F = [0]*(n+1)
    F[0] = 0
    F[1] = 1
    for i in range(2, n+1):
        F[i] =
    return F
```

Solution de l'exercice III.7 -

```
def encadrement(liste, a, b):
    n = len(liste)
    for i in range(n):
        if liste[i] < a or liste[i] > b:
            return False
    return True
```

Solution de l'exercice III.8 -

```
def liste_binom(n):
    binom = [0]*(n+1)
    binom[0] = 1
    for p in range(n-1):
        binom[p+1] = binom[p]*(n-p)//(p+1)
    return binom
```

Solution de l'exercice III.9 -

```
def somme_poly(P, Q):
    np = len(P)
    nq = len(Q)
    n1 = min(np, nq)
    n2 = max(np, nq)
    R = [0]*n2
    for i in range(n2):
        if i < n1:
            R[i] = P[i] + Q[i]
        elif np < nq:
            R[i] = Q[i]
        else:
            R[i] = P[i]
    return R
```

Solution de l'exercice III.10 -

```
def nb_quais(arrivees, departs):
    n = len(arrivees)
    ia = 0
    id = 0
    nb = 0
    nb_max = 0
    while ia < n:
        if arrivees[ia] < departs[ib]:
            nb = nb + 1
            ia = ia + 1
            if nb > nb_max:
                nb_max = nb
        else:
            nb = nb - 1
            ib = ib + 1
    return nb_max
```