EXERCICES

1. Fonction : définition et appel

Écrire une fonction `f` définie comme suit : $f(x) = 3x^2 - 1$. Dans le programme principal, vérifier que $f(0) = -1$, $f(1) = 2$ et $f(5) = 74$.

2. Fonction Celsius -> Fahrenheit

Écrire une fonction `C2F(C)` qui prend en paramètre une température C (en degrés Celsius) et qui renvoie sa conversion F (en degrés Fahrenheit), suivant la formule

$$F = C * 1,8 + 32$$

Combien font $42°C$ et $-273,15°C$ en $°F$?

3. Fonction pour année bissextile

Écrire une fonction `bissextile` qui prend en paramètre un entier `a` et retourne `True` si l'année `a` est bissextile (`False` sinon). Pour rappel, une année est bissextile si elle est soit multiple de 4 mais pas de 100, soit multiple de 400. Les années 2020 et 2038 sont elles bissextiles ?

4. Afficher un mouvement

Soit une liste `L` formée d'entiers parmi -1, 0 ou 1, par exemple `L= [-1, 0, -1, 0, 0, 1, -1, 0]`. Chaque entier représente une direction de déplacement :

— -1 représente le déplacement *à gauche*

— 0 représente le déplacement *tout droit*

— 1 représente le déplacement *à droite*

Ecrire une fonction `afficherDirection` qui prend en paramètre une liste `L` formée d'entiers parmi -1, 0 ou 1 et qui affiche une suite de directions de déplacement.

Par exemple, si `L= [-1, 0, -1, 0, 0, 1, -1, 0]` alors la fonction affichera :

```
à gauche
tout droit
à gauche
tout droit
tout droit
à droite
à gauche
tout droit
```

L'utilisation d'une liste est encouragée mais il est toutefois possible de répondre à l'exercice sans y avoir recours

5. Implémenter la fonction `max`

Écrire une fonction `max` qui renvoie le maximum de deux nombres `a` et `b` passés en paramètre.

6. Fonction binomiale

(1) Écrire une fonction `factorielle` qui renvoie la valeur factorielle de `n` où `n` étant passé en paramètre. Par exemple $1! = 1$ et $5! = 120$. On convient que $0! = 1$. On écrira une version de la fonction factorielle utilisant une boucle `for`.

(2) On rappelle que si n et p sont des entiers positifs ou nuls, le coefficient binomial est défini par

$$\binom{n}{p} = \begin{cases} \dfrac{n!}{p!(n-p)!} & \text{si} 0 \leq p \leq n \\ 0 & \text{sinon} \end{cases}$$

Ecrire une fonction `binomial(n, p)` qui renvoie le coefficient binomial ci-dessus.

3. Ecrire une fonction `maxBinomial(n)` qui renvoie la valeur maximale de tous les coefficients binomiaux $\binom{n}{0}, \binom{n}{1}, \binom{n}{2}, \cdots, \binom{n}{n}$

4. Ecrire une fonction `verifMaxBinomial(n)` qui vérifie que la valeur maximale de tous les coefficients binomiaux $\binom{n}{0}, \binom{n}{1}, \binom{n}{2}, \cdots, \binom{n}{n}$ est

$$\binom{n}{m}$$

où m est le quotient la division entière de n par 2.

7. Minimum et maximum simultanés

Ecrire une fonction `min_max` qui prend en paramètre une liste L et retourne sous forme de liste la plus petite et la plus grande valeur de L.

Exemples :

```
[12, 81, 65, 9, 32] -> [9, 81]
[42] -> [42, 42]
[42, 42, 42, 42, 42] -> [42, 42]
```

8. Plus petit entier non nul (découpage en fonctions)

1. Écrire une fonction `mini(L, i)` qui prend en paramètre
   - une liste `L` d'entiers
   - un indice valide positif `i` de la liste `L`

   et renvoie le plus petit des éléments de la liste `L` et ayant un indice supérieur ou égal à `i`.
   Voici quelques exemples de comportement de la fonction :

```
[40, 10, 50, 20] 0 -> 10
[40, 10, 50, 20] 1 -> 10
[40, 10, 50, 20] 2 -> 20
[40, 10, 50, 20] 3 -> 20
```

2. Écrire une fonction `plusPetitNonNul(L)` qui prend en paramètre une liste `L` d'entiers *non tous nuls* et qui renvoie le plus petit indice d'un élément non nul de `L`.
   Voici quelques exemples de comportement de la fonction :

```
[50, 10, 0, 90] -> 0
[0, 0, 70, 0, 42] -> 2
[0, 0, 70] -> 2
[42] -> 0
```

3. En utilisant les deux fonctions précédentes, écrire une fonction `miniNonNul(L)` qui prend en paramètre une liste `L` d'entiers *non tous nuls* et qui renvoie le plus petit élément *non nul* de `L`.

Voici quelques exemples de comportement de la fonction :

```
[50, 0, 0, 90, 10] -> 10
[0, 80, 10, 60, 0] -> 10
[0, 0, 0, 40, 0, 30, 0] -> 30
[0, 0, 0, 40, 0, 50, 0] -> 40
[0, 0, 0, 40, 0, 0, 0] -> 40
[0, 0, 0, 0, 0, 0, 40] -> 40
[50, 80, 10, 60, 40] -> 10
[42] -> 42
```

9. Formule de Keith et Craver

La formule de Keith et Craver permet de déterminer le jour de la semaine (ie lundi, mardi, etc) correspondant à une date donnée (par exemple, le 14 juillet 1789 qui était un mardi).

Ci-dessous, en voici une implémentation sous forme de fonction Python. La fonction `kc` retourne sous forme d'entier (`1 = lundi, 2 = mardi, ..., 7 = dimanche`) le jour de la semaine correspondant à une date passée en paramètre comme suit : `j` (jour), `m` (mois) et `a` (année).

```
1 def kc(j, m, a):
2     z = a - (m<3)
3     return (j + 23*m//9 + 3 -2*(m>=3) + a + z//4 - z//100 + z//400)%7 +1
```

Voici un exemple d'utilisation :

```
1 def kc(j, m, a):
2     z = a - (m<3)
3     return (j + 23*m//9 + 3 -2*(m>=3) + a + z//4 - z//100 + z//400)%7 +1
4
5 # Test du jour de la semaine du 1er septembre 2016
6 print(kc(1, 9, 2016))
7 4
```

— Lignes 6 et 7 : le 1er septembre 2016 est un jeudi (4e jour de la semaine).

Cette fonction doit être utilisée telle quelle, sans chercher à comprendre comment elle fonctionne. Vous devez réécrire cette fonction pour pouvoir l'utiliser par la suite mais il est inapproprié de copier/coller le corps de la fonction lignes 2 et 3 dans votre propre code. Il est seulement attendu d'**utiliser** la fonction `kc`.

(1) Vérifier les dates suivantes :

- vendredi 13 janvier 2016
- mardi 14 juillet 1789
- dimanche 10 mai 1981
- jeudi 16 juillet 1998
- lundi 1er mai 2017
- mardi 19 janvier 2038 (le bug de l'an 2038)

(2) En utilisant un appel à la fonction `kc`, écrire une fonction `est_vendredi13(m,a)` qui renvoie True si le 13 du mois m et de l'année a est un vendredi (et False sinon). Combien y-a-t-il de vendredis 13 dans l'année 2017 ?

3. Ecrire et tester une fonction `jour_semaine` qui accepte en argument une **liste** `date`, supposée de la forme `[jour, mois, année]` et qui retourne le jour de la semaine correspondant en toutes lettres (ex : *lundi*, *mardi*, .., *dimanche*).

Par exemple, `jour_semaine(14, 7, 1789)` vaut `mardi`.

10. Calcul d'une durée

1. Écrire une fonction `hms_to_s` qui retourne en secondes une durée exprimée en heures, minutes et secondes. Vérifier que 1h30m = 5400s et 3h20m15s = 12015s.
2. Écrire une fonction `s_to_hms` qui prend en paramètre une durée exprimée en secondes et la retourne exprimée en heures, minutes et secondes sous forme de liste [h, m, s]. Vérifier que `s_to_hms(5400)` retourne la liste `[1, 30, 0]`, et que `s_to_hms(12015)` retourne la liste `[3, 20, 15]`.
3. Écrire une fonction `afficher_temps` capable d'afficher une liste [h, m, s] sous la forme `h heures m minutes et s secondes`. Par exemple, `afficher_temps([2, 42, 16])` affiche `2 heures 42 minutes et 16 secondes`.
4. Écrire un programme qui affiche, en une seule ligne de code, le temps écoulé entre 13h58m et 15h31m30s.

11. Liste croissante

Ecrire une fonction `est_croissante` qui détermine si une liste d'entiers passée en paramètre est croissante (i.e. renvoie True) et False sinon.

Qu'une liste `L` soit croissante signifie que $L[i] \leq L[i+1]$ pour tout indice `i` pour lequel l'inégalité a un sens.

Exemples :

```
[5, 6, 6, 6, 10] -> True
[5, 6, 6, 6, 4] -> False
[42] -> True
```

12. Aucun multiple de 10

Écrire une fonction `aucunMultiple10` qui prend en paramètre une liste d'entiers et qui renvoie True si la liste ne contient aucun multiple de 10. Voici deux exemples de comportement de la fonction :

```
[42, 81, 33, 81] -> True
[42, 91, 75, 90, 33] -> False
```

13. Remplacer les nombres négatifs

Ecrire une fonction `rempl_neg` qui prend en paramètre une liste d'entiers L, ne retourne rien mais remplace dans la liste L chaque entier négatif en son opposé, par exemple :

```
[-6, 2, 5, -3, 0, 2, 0, -1] -> [6, 2, 5, 3, 0, 2, 0, 1]
```

Ainsi, la fonction `rempl_neg` **modifie** la liste qu'elle reçoit en argument.

14. Un élément sur trois

Ecrire une fonction f qui prend une liste `L` d'entiers et qui renvoie la liste `M` formée d'un élément sur 3 de la liste `L` en commençant par la fin. Exemples :

```
[19, 43, 25, 54, 71, 76, 88, 53, 67, 22, 90, 79, 12, 25] -> [25, 90, 53, 71, 43]
[19, 43, 25, 54] -> [54, 19]
[19] -> [19]
```

15. Liste sans répétition successive

Soit, par exemple, la liste L d'entiers :

`[3, 7, 5, 5, 8, 8, 8, 8, 2, 5, 5, 5]`.

Elle contient des répétitions *successives* comme `8, 8, 8, 8` ou encore `5, 5, 5`. La liste sans répétition successive est `[3, 7, 5, 8, 2, 5]`.

Plus généralement, écrire une fonction `f(L)`, où `L` est une liste non vide d'entiers, qui renvoie la liste `M` formée des éléments de `L` sans ses répétitions de termes successifs. La liste `M` doit être une *nouvelle* liste et la liste `L` ne doit pas être modifiée par la création de `M`.

Exemples :

```
[7, 7, 5, 5, 8, 8, 8, 8, 5, 5, 5] -> [7, 5, 8, 5]
[7, 7, 7] -> [7]
[7, 0, 1] -> [7, 0, 1]
[7] -> [7]
```

16. ppcm

Ecrire une fonction `ppcm` qui renvoie le plus petit commun multiple de deux entiers donnés. Par exemple, `ppcm(16, 60)` vaut 240.

Il est attendu d'écrire une version « naïve » (et non pas d'utiliser l'algorithme d'Euclide pour obtenir d'abord le pgcd). La version naïve consistera par exemple à examiner les multiples non nuls de `a` et à tester jusqu'à ce qu'un de ces multiples soit multiple de `b`.

17. Suite de Syracuse

(1) Soit la fonction f définie pour $n > 0$ entier par :

- $f(n)$ est la moitié de n si n est pair
- $f(n)$ vaut $3n + 1$ si n est impair

Par exemple `f(13) = 40` ou encore `f(10) = 5`.
Coder en Python cette fonction. Tester avec $f(13)$ et $f(10)$.

(2) On itère la fonction f en partant d'un entier $n > 0$. Par exemple, si $n = 13$, on obtient :

```
13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

La conjecture de Syracuse affirme que si on itère f depuis n'importe quel entier $n > 0$ alors, on tombera forcément sur 1 (ce qui se produit bien pour l'exemple ci-dessus).
Écrire une fonction `seq` qui

- prend n en paramètre,
- affiche les itérés de la suite depuis le premier itéré, à savoir n jusqu'au dernier, à savoir 1.

(3) Écrire une fonction `saut(x)` qui renvoie le nombre d'itérations pour arriver à 1 quand la suite commence avec x (si $x = 13$, il y a donc 9 itérations pour arriver à 1).

(4) Déterminer à l'aide de la fonction fonction `saut(x)` le plus petit entier n tel qu'il faille exactement 100 itérations à partir de n pour arriver à 1 (vous devez trouver $n = 107$).

18. Le tri par sélection

Le *tri par sélection* est un algorithme de tri, utilisé par exemple pour trier une liste de nombres dans l'ordre croissant. Son principe est le suivant : on dispose d'une liste `t`, on en cherche le plus grand élément `m`, on le met « de côté », on recommence avec la liste restante en recherchant à nouveau son plus grand élément, puis on le place à gauche de `m`, et ainsi de suite jusqu'à ce que la liste initiale soit vide. La liste triée est alors la liste construite avec les maxima successifs.

Quand on programme un tri par sélection, pour éviter de multiples suppressions/recopies, au lieu de mettre « de côté » les maxima trouvés, on les déplace à l'intérieur même de la liste `t`. Plus précisément, soit une liste `t` de `n` entiers. Le tri par sélection consiste à :

— chercher le plus grand élément de `t`,

— l'échanger avec l'élément le plus à droite de `t`

— recommencer avec la liste formée des $n-1$ premiers éléments de `t`.

(1) Ecrire une fonction `maxi(t, p)` qui prend en paramètre une liste d'entiers `t` de longueur `n`, et un indice `p` valide de `t` (donc tel que $0 \leq p \leq n-1$) et qui renvoie l'**indice** d'un élément maximum de la sous-liste de `t` qui s'étend de l'indice `0` (inclus) à l'indice `p` (inclus).

(2) (a) Ecrire une procédure `echange(t, i, j)` qui prend en paramètre une liste d'entiers `t` et deux indices `i` et `j` d'éléments de `t` et qui échange les contenus dans `t` aux indices `i` et `j`.

(b) Écrire le code d'un tri par sélection avec une procédure `selection(t)` et qui utilise `maxi` et `echange`.

(c) Générer une liste `L` de 30000 entiers et lui appliquer la fonction `selection`. Que pensez-vous de la vitesse d'exécution ?

19. Tirage du loto

Écrire une fonction `tirerLoto` qui génère un tirage aléatoire des 6 numéros d'un loto.

Rappel : si `L` est une liste d'entiers et `x` un entier alors l'expression `x in L` vaut `True` si l'entier `x` est dans la liste `L` et `False` sinon.

20. PILE quatre fois de suite

(1) Ecrire une fonction `lancer_piece` qui simule le lancer d'une pièce de monnaie équilibrée. La fonction retournera la chaîne "PILE" ou la chaîne "FACE".

(2) On dispose d'une pièce de monnaie. On cherche à savoir le minimum de lancers qu'il faut effectuer pour obtenir 4 fois *FACE* consécutivement
Créer une fonction `attente` qui effectue une suite de lancers de la pièce et qui renvoie le nombre de lancers qu'il faut faire pour obtenir pour la première fois 4 fois FACE de suite.

(3) Ecrire une fonction `test` qui itère `n` l'expérience de la fonction `attente` la fonction `attente`. Vérifier qu'il faut en moyenne 30 lancers pour obtenir 4 fois de suite FACE.

Tester sur $n = 10^5$ expériences.

21. Jeu du 421

Le jeu du 421 consiste à obtenir avec 3 dés une combinaison contenant une fois le chiffre 4, une fois le chiffre 2 et une fois le chiffre 1. Par exemple, si vous jetez trois dés simultanément et que vous obtenez la combinaison 2, 1 et 4, vous avez gagné ; si vous obtenez 1, 4, 1 alors votre coup n'est pas gagnant.

L'exercice va demander combien de jets de 3 dés sont nécessaires, en moyenne, pour faire un 421.

(1) Ecrire une fonction `est421` qui prend en paramètre une liste de trois entiers et renvoie

— `True` si la liste, à la fois, contient 4, contient 2 et contient 1

— `False` sinon.

Ainsi :

— `est421([6, 2, 1]) = False`

— `est421([1, 1, 1]) = False`

— `est421([2, 4, 1]) = True`

— `est421([1, 2, 4]) = True`

— `est421([4, 2, 1]) = True`

(2) Ecrire une fonction `tirage421` qui renvoie, sous forme de liste, un tirage aléatoire de trois dés. Par exemple, si vous tirez les nombres

```
6, 2, 2
```

alors la fonction doit renvoyer la liste `[6, 2, 2]`. Les nombres peuvent avoir été tirés simultanément ou l'un après l'autre, c'est indifférent.

(3) Ci-dessous, on appelle *suite gagnante* toute succession de lancers de trois dés, autant de fois que nécessaire, pour « sortir » 421. Par exemple, la suite de 7 lancers suivants est une suite gagnante :

```
523 144 643 235 451 252 214
```

Écrire une fonction `nombreSuitesGagnantes` qui ne prend aucun paramètre et qui

— simule une suite gagnante,

— renvoie le nombre de lancers de la suite gagnante

Par exemple, `nombreSuitesGagnantes` renvoie 7 dans le cas de la suite gagnante ci-dessus.

(4) On fait 1000 suites gagnantes. Calculer le nombre moyen de lancers d'une suite gagnante (le résultat théorique est 36).

22. Liste monotone

Une liste d'entiers est monotone ou bien si elle est croissante ou bien décroissante.

Ecrire une fonction `est_monotone` qui determine si une liste d'entiers est monotone (i.e. renvoie True et False sinon).

Il pourra être utile de remarquer que deux nombres ont des signes différents si leur produit est strictement négatif.

Exemples :

```
[5, 5, 6, 6, 6, 10] -> True
[5, 5, 6, 6, 6, 6] -> True
[5, 5, 6, 6, 6, 5] -> False
[42, 5, 6, 6, 6, 5] -> False
[42] -> True
[42, 42, 42, 42, 42] -> True
[5, 4] -> True
[4, 5] -> True
[5, 4, 3, 3, 0] -> True
[5, 4, 3, 42, 43, 44] -> False
[5, 4, 3, 3, 42] -> False
[] -> True
```

23. Supprimer les doublons

Partant d'une liste d'entiers, définir une variable `sansDoublons` qui référence une liste composée des valeurs de L, mais dans laquelle les différentes valeurs égales de L (s'il y en a) n'ont été copiées qu'une seule fois. On dira ainsi de la liste construite qu'elle ne contient pas de doublon. Exemples :

```
[42, 81, 42, 65, 12, 81, 31, 42] -> [42, 81, 65, 12, 31]
[42] -> [42]
[42, 42, 42, 42, 42] -> [42]
```

24. Fractions

Dans cet exercice, une fraction sera représentée par une liste de 2 entiers positifs :

`[numérateur, dénominateur]`.

Par exemple, la fraction $\frac{22}{7}$ sera représentée par la liste `[22, 7]`. Plus généralement, la fraction $\frac{a}{b}$ sera représentée par la liste `[a, b]`.

1. Écrire une fonction `aff` qui affiche une fraction `frac` donnée en paramètre. Par exemple, `aff([22,7])` affichera `22 / 7`.
2. Écrire une fonction `add(frac1, frac2)` qui retourne la somme des fractions `frac1` et `frac2`. On appliquera la formule suivante : $\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$. Par exemple, `add([1,3],[7,10])` retournera `[31,30]`. On ne cherchera pas à obtenir une fraction simplifiée (autrement dit une fraction irréductible).
3. Écrire une fonction `harmonique(n)` qui calcule la somme $1+\frac{1}{2}+...+\frac{1}{n}$. Vérifier que `harmonique(6)` affiche `1764 / 720`.
4. Ecrire une fonction `pgcd(a, b)` qui retourne le plus grand diviseur commun de deux entiers a et b. Par exemple, pgcd(140, 100) = 20. On procédera de la manière naïve suivante : parcourir tous les entiers d entre 1 et, par exemple a, et stocker d dans une variable chaque fois que, simultanément, a est multiple de d et que b est multiple de d.
5. Écrire une fonction `simplifier(frac)` qui simplifie la fraction `frac` reçue en paramètre. Par exemple, `simplifier([140, 100])` doit renvoyer `[7,5]`.

Si vous essayez de simplifier `harmonique(12)`, vous allez constater que l'exécution va être extrêmement longue.

Quitte à récrire légèrement le code de la fonction `add`, parvenir à simplifier `harmonique(12)`

25. Vendredi 13

On se propose d'écrire un code qui recherche les 10 prochains vendredis 13.

On codera chaque mois par un entier entre 1 et 12, chaque jour par un entier entre 1 et 31 et chaque jour de la semaine par un code (lundi = 1, mardi = 2, ..., dimanche=7).

1. Ecrire une fonction `lendemain(js, j, m, a)` qui prend une date en paramètres et renvoie une liste définissant la date du lendemain. Par exemple `lendemain(2, 8, 12, 2015)` demande la date du lendemain du mardi 8 décembre 2015 et doit renvoyer la liste `[3, 9, 12, 2015]` car le lendemain est le mercredi 9 décembre 2015.
   Les paramètres ont les significations suivantes :
   - le jour de la semaine `js` (un entier entre 1 et 7)
   - le numéro `j` du jour dans le mois (un entier entre 1 et 31)
   - le numéro `m` du mois (un entier entre 1 et 12)

— l'année `a`.

La fonction traitera successivement les cas suivants :

— le 31 décembre,

— la fin des mois autres que décembre et ayant 30 ou 31 jours,

— la fin du mois de février,

— tous les autres jours.

On pourra utiliser les listes suivantes :

— `mois30 = [4, 6, 9, 11]`

— `mois31 = [1, 3, 5, 7, 8, 10]`

sachant qu'on peut tester l'appartenance d'un objet `x` dans une liste `L` par :

```
x in L
```

On rappelle qu'une année `a` est bissextile si le booléen

```
a % 400 == 0 or (a % 4 == 0 and a % 100 != 0)
```

est `True`.

Pour trouver le jour de la semaine du lendemain, on pourra utiliser un reste de division par 7 mais ce n'est pas indispensable, on peut toujours faire une instruction `if`.

On effectuera les tests suivants :

```
(2, 8, 12, 2015) -> [3, 9, 12, 2015]
(5, 1, 1, 2016) -> [6, 2, 1, 2016]
(4, 31, 12, 2015) -> [5, 1, 1, 2016]
(7, 31, 1, 2016) -> [1, 1, 2, 2016]
(6, 30, 1, 2016) -> [7, 31, 1, 2016]
(1, 29, 2, 2016) -> [2, 1, 3, 2016]
(7, 28, 2, 2016) -> [1, 29, 2, 2016]
(6, 28, 2, 2015) -> [7, 1, 3, 2015]
(6, 16, 7, 2016) -> [7, 17, 7, 2016]
```

(2) Déterminer les 10 prochains vendredis 13 à partir d'aujourd'hui. Vérifiez avec un calendrier.