ESIR

CUPGE 1

Algorithmique

P. Rannou

2021-2022

Travaux Dirigés

1 Instructions de base

Exercice 1

1.1. Donner les valeurs des variables x, y et z à la fin de l'exécution du programme suivant :

```
x = 4
y = 2+x
z = x-y
y = y+z
```

1.2. Échange de deux valeurs

On suppose que deux variables `a` et `b` ont déjà été initialisées. On note a_i la valeur initiale de `a` et b_i la valeur initiale de `b`.

a. Donner les valeurs de `a` et de `b` après la séquence d'instructions suivante :

```
c = a
a = b
b = c
```

b. Quelles seraient les valeurs de `a` et de `b` si à la place on avait effectué la séquence d'instructions suivante :

```
a = b
b = a
```

Exercice 2

2.1. Donner le type (`int`, `float`, `bool` ou `str`) de chacune des valeurs suivantes :

a. `4.5` **b.** `'4'` **c.** `False` **d.** `0`

2.2. Donner le type (`int`, `float`, `bool` ou `str`) de chacune des expressions suivantes. Préciser aussi la valeur de cette expression.

a. `float(4)`
b. $4 > 5.5$
c. $(4 + 5.5) * 2$
d. `str`$(4 + 5)$
e. `str`$(4) +$ `str`(5)
f. $((3 < 4)$ and $(4 < 3))$ or $3 \neq 4$
g. $9/3$
h. $7//$`int`(3.14)
i. `'zebre'<'lion'`

2.3. On dispose d'une variable entière `n` prédéfinie.

On souhaite créer une variable `t` de type `str` précisant le double de `n` : si par exemple `n` vaut 4 alors `t` doit valoir :

```
'Le double de 4 est 8.'
```

Attention : il s'agit de créer une nouvelle variable, et pas d'afficher un message avec `print`.

Exercice 3

3.1. Le savant Sunisoc a écrit un programme permettant avec une instruction `input` de rentrer un nombre et d'afficher son double. Le programme est donné ci–dessous.

```
a = input("Rentrez un nombre entier : ")
print ("Le double de ce nombre est : ", a+a)
```

Il y a une erreur dans ce programme. Si un utilisateur écrit ce programme et rentre la valeur 4, quel affichage surprenant obtiendra–t–il ?

Exercice 4 Intersection de deux rectangles

Le plan est rapporté à un repère orthonormé direct $(O, \vec{\imath}, \vec{\jmath})$.

On se donne deux rectangles R_1 et R_2 dont les côtés sont parallèles aux axes de coordonnées.

Soient A le sommet inférieur gauche de R_1 et B son sommet supérieur droit.

Soient C le sommet inférieur gauche de R_2 et D son sommet supérieur droit.

On suppose connues les coordonnées `xA`, `yA`, `xB`, `yB`, `xC`, `yC`, `xD`, `yD` de ces 4 sommets.

4.1. Définir un test permettant de savoir si ces deux rectangles s'intersectent.

Exercice 5

5.1. Donner dans chacun des cas la valeur de la variable `a` après l'exécution du programme :

a.
```
a = 4
if a*2 > 10 :
    a = a-8
else :
    a = a+8
```

b.
```
a=int(input("Rentrez un entier"))
a=a*2
a=a**2
a=a**3
```

c.
```
a = 5
if a < 4 :
    a += 2
a -= 2
```

d.
```
a = 5
if a < 4 :
    a += 2
    a -= 2
```

e.
```
a = 5
if a%2 == 0 :
    a += 3
else :
    a //= 2
```

f. *Traiter les différents cas*

```
b=int(input("Rentrez un entier"))
a=2*b+4
if a == b :
    a = 0
```

Exercice 6 Imbrication d'embranchements conditionnels

Une compagnie de bus propose des tarifs de groupe pour un trajet :

- 10€ le ticket pour un seul passager
- 8€ le ticket s'il y a entre 2 et 3 passagers
- 7€ le ticket s'il y a entre 4 et 5 passagers
- 6€ le ticket s'il y a 6 passagers ou plus

6.1. Définir un algorithme demandant à l'utilisateur d'entrer le nombre de personnes dans un groupe et affichant le prix **total** que paiera ce groupe pour le trajet.

Un informaticien propose de coder ce calcul en `python` de la manière suivante :

```
if passagers < 4 :
    if passagers >= 2 :
            prix = passagers * 8
    else :
            prix = 10
else :
    if passagers < 6 :
            prix = passagers * 7
    else :
            prix = passagers * 6
```

6.2. Est-ce que son code est correct ?

6.3. Donner une autre façon de calculer ce prix en utilisant des `elif`.

Exercice 7

7.1. Préciser dans chacun des cas suivants l'éventuel message affiché par le programme (ou un message d'erreur si le programme est incorrect) :

a.
```
a = 6
t = (a%2 == 0)
if True == t :
    print ("Le nbr",a,"est pair")
```

b.
```
a = 6
t = (a%2 == 0)
if t :
    print ("Le nbr",a,"est pair")
```

c.
```
a = 6
if a%2 :
    print ("Le nbr",a,"est pair")
```

d.
```
a = 6
t = (a%2 == 0)
if t == True :
    print ("Le nbr',a,'est pair")
```

Exercice 8 Programme mystère

8.1. Préciser en fonction de l'entier `n` la valeur de `y` après l'exécution du programme suivant. On précisera les différents cas.

```
if n >= 3:
    y = n*n
    if n <= 4 :
        y = y-1
        y = 2*y
    else :
        y += 1
else :
    if n % 2 == 0 :
        y = 3*n
    else :
```

```
        y = n-5
    y = 3*y
```

Exercice 9 Minimum de trois valeurs

On suppose qu'on a déjà défini trois variables réelles `a`, `b` et `c`.

9.1. Écrire une conditionnelle utilisant uniquement des tests $<$ qui permet d'affecter à la variable `d` la plus petite des trois valeurs entre `a`, `b` et `c`
(remarque : il existe une solution plus simple : `d=min(a,b,c)`*).*

Exercice 10

10.1. On suppose qu'une variable `a` est entrée au clavier par l'utilisateur. Qu'affiche l'algorithme suivant ?

```
a = int(input())
b = 0
while b*b < a :
    b = b+1
print(b)
```

Exercice 11

On définit une suite (u_n) par $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = \cos(u_n)$.
Un entier naturel n est lu au clavier et stocké dans une variable `n` (noter la différence entre l'entier mathématique n écrit en italique et qui ne change pas, et la variable informatique `n` écrite en police à chasse fixe).

11.1. À la fin de l'exécution de l'algorithme suivant, la variable `u` contient–t–elle u_n, u_{n+1} ou autre chose ?

```
n = int(input())
u = 0
while n > 0 :
    u = cos(u)
    n = n-1
```

Exercice 12

La suite de Fibonacci (F_n) est définie par $F_0 = 0$, $F_1 = 1$ et $\forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n$.
Un entier naturel n est lu au clavier et stocké dans une variable `n`.

12.1. À la fin de l'exécution de l'algorithme suivant, la variable `f` contient–t–elle F_n, F_{n+1} ou autre chose ? Et la variable `b` ?

```
n = int(input())
f = 0
a = 1
while n > 0 :
    b = a+f
    f = a
    a = b
    n = n-1
```

Exercice 13

Le programme suivant permet de calculer le plus grand diviseur impair d'un entier positif n :

```
n = int(input())
while n%2 == 0 :
    n = n//2
print(n)
```

13.1.

a. Que se passe–t–il si la valeur initialement rentrée dans `n` est 0 ?

b. Pourquoi le programme précédent est–il dangereux ?

c. Comment le modifier pour éviter les risques de "plantage" de l'ordinateur ?

Exercice 14

14.1. Écrire un algorithme utilisant une boucle `while` qui lit un entier n au clavier et qui affiche la somme des entiers naturels k tels que $k^2 + 3k <$ `n`.
Par exemple si `n` = 20 on doit trouver $0+1+2+3 = 6$ car 3 est le plus grand entier naturel pour lequel $k^2 + 3k < 20$.

14.2. Serait–il possible de résoudre ce problème sans utiliser de boucle `while` ?

Exercice 15 Punition

L'instruction `print ("Je dois ranger ma chambre")` provoque l'affichage du message :

```
Je dois ranger ma chambre
```

15.1. Écrire un programme avec une boucle qui affiche 50 fois le message `Je dois ranger ma chambre`.

Exercice 16 Chanson traditionnelle bretonne

La séquence d'instructions :

```
n = 10
print ("C'est dans", n, "ans je m'en irai j'entends le loup le renard chanter")
```

permet d'afficher le message :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
```

16.1. Écrire une boucle `python` qui permet d'afficher :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
C'est dans 9 ans je m'en irai j'entends le loup le renard chanter
C'est dans 8 ans je m'en irai j'entends le loup le renard chanter
...
C'est dans 1 ans je m'en irai j'entends le loup le renard chanter
```

On ne s'occupera pas de la faute d'orthographe de la dernière ligne.

Exercice 17

17.1. Écrire un algorithme lisant un entier `n` au clavier et affichant $H_{\texttt{n}} = \prod_{k=1}^{\texttt{n}} k^k$.

17.2. Écrire un algorithme lisant un entier `n` au clavier et affichant $\sum_{i=1}^{\texttt{n}} H_i$.

Exercice 18

18.1. Écrire un algorithme demandant à l'utilisateur d'entrer un entier n au clavier et permettant de calculer le n–ième terme de la suite (u_n) définie par

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = u_n + \cos(u_n) \end{cases}$$

18.2. Écrire un algorithme demandant à l'utilisateur d'entrer un entier n au clavier et permettant de calculer le n–ième terme de la suite (u_n) définie par

$$\begin{cases} u_0 = 1 \\ u_1 = 0 \\ \forall n \in \mathbb{N}, u_{n+2} = u_{n+1} + \sin(u_n) \end{cases}$$

Pour aller plus loin

Exercice 19

On modélise une population de bactéries dans une solution de culture de la façon suivante : chaque jour, chaque bactérie :

- ou bien consomme une unité de nourriture, elle libère alors une quantité de toxine égale à $(t+1)e^{-t}$ où t est la quantité de toxine présente au début de la journée, puis elle se duplique ;
- ou bien elle meurt sans produire de toxine s'il n'y a plus de nourriture disponible.

Initialement il y a une seule bactérie, un nombre (entier) d'unité de nourriture mémorisé dans une variable `n`, et une quantité nulle de toxine.

19.1. Concevoir un algorithme permettant de calculer la quantité totale de toxine produite.

Exercice 20 Le problème de Josephus

Cette histoire est issue de la vie de l'historien antique Flavius Josèphe, qui a vécu sous l'empereur romain Vespasien (même si sa véracité n'est pas garantie).

Soient $k, n \in \mathbb{N}^*$. n personnes sont positionnées en cercle, numérotées de 0 à $n-1$. On élimine itérativement un survivant sur k en tournant, jusqu'à ce qu'il n'en reste plus qu'un (ainsi, lorsque $k \leqslant n$, la première personne éliminée a pour numéro $k-1$). Le but est de déterminer le numéro $S(n,k)$ du dernier survivant.

20.1.

a. Démontrer que $\begin{cases} \forall n \geqslant 2, S(n,k) \equiv S(n-1,k) + k \bmod n, \\ S(1,k) = 0. \end{cases}$

Pour la première identité on pourra par exemple raisonner sur le numéro j de la première personne éliminée, puis se ramener au problème de Josephus à $n-1$ personnes.

b. En déduire un programme permettant de calculer $S(n,k)$.

Exercice 21

En 2004 on a découvert que le nombre $n = 28433 \times 2^{7830457} + 1$ est un nombre premier (son écriture décimale est composée de 2 357 207 chiffres). Ce nombre est beaucoup trop grand pour pouvoir être calculé en temps raisonnable par l'interpréteur Python.

21.1. Comment calculer les 10 derniers chiffres de son l'écriture décimale en temps raisonnable ?

Exercice 22 La constante de Champernowne

La constante de Champernowne est obtenue en concaténant derrière la virgule (ou le point) la suite des écritures décimales des entiers consécutifs :

$$0.12345678910111213141516171819202 1\ldots$$

On note d_n la n–ième décimale de ce nombre : par exemple $d_1 = 1$ et $d_{15} = 2$.

22.1. Écrire un algorithme permettant de calculer d_n.

Exercice 23

Il est classique de coder un rationnel $\frac{p}{q}$ par le couple d'entiers $(\mathtt{p}, \mathtt{q})$.

Dans cet exercice on définit la fonction $f : \mathbb{N} \to \mathbb{Q}^+$ définie par :

$$f(0) = 0 \text{ et } \forall n \in \mathbb{N}^*, f(2n) = \frac{1}{f(n)+1} \text{ et } \forall n \in \mathbb{N}, f(2n+1) = f(n) + 1$$

et on admettra que f est une bijection de $\mathbb{N}$ sur $\mathbb{Q}^+$.

23.1. Écrire la fonction `f(n)` qui renvoie la valeur de $f(n)$ sous forme d'un couple d'entiers.

Remarque : si vous connaissez la récursivité, c'est plus facile à programmer, mais on peut s'en passer.

23.2. On note $g = f^{-1}$ la bijection réciproque. Programmer cette fonction `g(p,q)`. On cherchera une solution plus efficace que de tester un à un tous les entiers...

2 Types de données structurés

Exercice 24

24.1. À la fin de ce code, quelle est la valeur de `len(a)` ?

```
a = "a"
a *= 2
a = a + a + "a"
```

24.2. Donner la valeur de l'expression :

```
('mon '+2*'dou'+' '+2*"nou"+'rs')[1:18]
```

Exercice 25 Recherche d'un mot dans un texte

On dispose d'une chaîne de caractères courte mémorisée dans une variable `mot` de type `str`, ainsi que d'une chaîne (probablement plus longue) de caractères mémorisée dans une variable `texte` de type `str` aussi. La longueur de `mot` est `p` et la longueur de `texte` est `n`.

On souhaite savoir si la chaîne `mot` se trouve à l'intérieur de la chaîne `texte` : plus formellement on souhaite savoir si :

$$\exists\ 0 \leqslant \texttt{i} \leqslant \texttt{j} \leqslant \texttt{n} \mid \texttt{mot} == \texttt{texte[i : j]}$$

On considère que si `a` et `b` sont deux chaînes de caractères de même longueur $\texttt{q} \in \mathbb{N}$, le test `a == b` nécessite pour l'interpréteur Python au plus la comparaison de `q` lettres.

25.1. Résoudre le problème posé avec une solution nécessitant au plus la comparaison de $\texttt{p} \times (\texttt{n} - \texttt{p} + 1)$ lettres.

25.2. Modifier l'algorithme précédent, de façon à renvoyer la liste des indices où `mot` apparaît dans `texte`. Plus précisément, renvoyer la liste des indices :

$$\texttt{i} \in [0, \texttt{n}] \text{ tels que } \exists \texttt{j} \in [\texttt{i}, \texttt{n}] \mid \texttt{mot} == \texttt{texte[i : j]}$$

Exercice 26

26.1. Générer la liste $\ell = [\ell_0, \ell_1, \ldots, \ell_{\texttt{n}-1}]$ avec $\ell_{\texttt{k}} = 2\texttt{k} + 1$.

26.2. Générer la liste $\ell = [\ell_0, \ell_1, \ldots, \ell_{\texttt{n}-1}]$ avec $\ell_{\texttt{k}} = \texttt{k}^2$.

Exercice 27

On définit une suite (u_n) par $u_0 = 2$ et $\forall n \in \mathbb{N}, u_{n+1} = u_n + \sqrt{u_n} + 1$.

27.1. Écrire un algorithme qui permet de calculer la liste $[u_0, u_1, \ldots, u_{n-1}]$.

Exercice 28 *List comprehension*

28.1. Quel est le résultat de l'expression suivante (essayer de deviner avant de l'écrire) :

```
[i**2+1 for i in range (5)]
```

Est–ce qu'on obtient la même liste dans la variable `res` si on exécute le programme suivant :

```
res = []
for i in range (5) :
    res.append (i**2+1)
```

Exercice 29

Le savant Sunisoc crée une liste de listes nommée `x` :

```
>>> x = [[3, 4], [5, 9]]
```

Il souhaite dupliquer cette liste de listes dans une variable nommée `y` de façon à ce que toute modification subséquente de `x` ne modifie pas `y`.
Il utilise donc le constructeur de listes :

```
>>> y = list (x)
```

Pourtant une surprise l'attend :

```
>>> x[1][1] = 10
>>> x
[[3, 4], [5, 10]]
>>> y
[[3, 4], [5, 10]]
```

29.1. Expliquer ce qui s'est passé et corriger son code pour arriver au résultat voulu.

Exercice 30

30.1. Écrire un programme qui permet de supprimer toutes les occurrences d'un élément dans un tableau. Par exemple si on part de la liste `l=[2,6,2,2,4,5,2,3]` et qu'on supprime toutes les occurrences de 2, à la fin du programme la liste `l` doit être égale à `[6,4,5,3]`.

30.2. En utilisant la fonction `len` et une *list comprehension*, écrire un programme qui calcule la liste miroir d'une liste donnée (c'est–à–dire la liste des éléments dans l'autre sens).

Tester : si la liste d'entrée est `[3,6,1,8,2,4,0]`, son miroir est `[0,4,2,8,1,6,3]`.

Exercice 31

Une liste est un *palindrome* si c'est la même qu'on la lise de gauche à droite ou de droite à gauche.

31.1. Écrire un algorithme permettant de savoir si une liste est un palindrome.

Tester : pour `[4,2,4,2]` on doit renvoyer `False`, pour `[4,2,4]` ça doit être `True`.

Exercice 32 Second plus petit élément

32.1. Écrire un algorithme qui, à partir d'une liste d'entiers *distincts*, calcule le second plus petit élément de cette liste.

Exercice 33

Un tableau à double entrée (matrice) à valeurs entières est codé par une liste de listes.

Par exemple la matrice $\begin{array}{|c|c|c|}\hline 2&3&7\\\hline 1&2&0\\\hline\end{array}$ est codée par :

$$[[2,3,7],[1,2,0]]$$

33.1. Écrire une fonction calculant le nombre de lignes d'une matrice. Sur notre exemple, la valeur de retour doit être 2.

33.2. Écrire une fonction calculant le nombre de colonnes d'une matrice. Sur notre exemple, la valeur de retour doit être 3.

33.3. Écrire une fonction renvoyant le nombre d'éléments impairs de la matrice. Sur notre exemple, la valeur de retour doit être 3.

33.4. Écrire une fonction renvoyant la liste des sommes des éléments de chaque colonne. Sur notre exemple, la valeur de retour doit être $[3,5,7]$.

33.5. Écrire une fonction renvoyant la liste des sommes des éléments de chaque ligne. Sur notre exemple, la valeur de retour doit être $[12,3]$.

Exercice 34

Un tableau `t` à double entrée à n lignes et n colonnes est dit symétrique si, lorsqu'on note $c_{i,j}$ le coefficient à la ligne i et à la colonne j : $\forall i,j \in [0,n-1], c_{j,i} = c_{i,j}$

34.1. Écrire une fonction dont le paramètre est le tableau `t` et qui renvoie un booléen indiquant si `t` est symétrique ou non.

Exercice 35

35.1. Écrire une fonction permettant de transposer un tableau à double entrée, c'est–à–dire que si `t` est un tableau à p lignes et q colonnes, on doit calculer le tableau à q lignes et p colonnes tel que pour tout $i \in [0,q-1]$, pour tout $j \in [0,p-1]$, l'élément de la ligne i et de la colonne j du tableau final est l'élément ligne j colonne i du tableau de départ.

Exercice 36 Élection : max **du** min

p électeurs $e_0,\dots,e_{p-1}$ participent à une élection. Il y a q candidats $c_0,\dots,c_{q-1}$. Le vote se déroule ainsi : chaque électeur e_i ($i \in [0,p-1]$) attribue une note $n_{i,j}$ à chaque candidat c_j ($j \in [0,q-1]$). Le candidat retenu est celui dont la note la plus basse (parmi les notes qui lui sont attribuées) est plus haute (parmi les différents candidats). Remarquer qu'un tel candidat n'est pas nécessairement unique. Dit autrement, un candidat c_j est retenu si :

$$\forall k \in [0,q-1], \min\{n_{0,j},\dots,n_{p-1,j}\} \geqslant \min\{n_{0,k},\dots,n_{p-1,k}\}$$

On mémorise dans un tableau à double entrée les notes $n_{i,j}$ attribuées (avec i en indice de ligne et j en indice de colonne).

36.1. Écrire une fonction `python` dont le paramètre est le tableau à double entrée des notes et dont la valeur de retour est la liste des candidats retenus.

36.2. Écrire une fonction `python ajout_colonne` dont les deux paramètres sont un tableau à double entrée `t` à p lignes, et une liste `c` de longueur p, et qui modifie le tableau `t` en rajoutant `c` comme une nouvelle colonne apparaissant à droite de `t`. Il n'y a pas de valeur de retour ; c'est le tableau `t` qui est modifié.

Exercice 37 Carré magique

Un *carré magique* est une matrice de n lignes et n colonnes, dont les coefficients sont les entiers appartenant à $[1, n^2]$ (chaque entier apparaissant une et une seule fois) et telle que les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales du carré sont toutes les mêmes.

Par exemple

4	9	2
3	5	7
8	1	6

est un carré magique : les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales sont toutes égales à 15, et chaque entier de $[1, 9]$ apparaît une et une seule fois.

37.1. Écrire une fonction qui permet de savoir si les sommes de chaque ligne, de chaque colonne et des deux diagonales sont toutes les mêmes.

37.2. Écrire une fonction qui permet de savoir si chaque élément de $[1, n^2]$ apparaît une et une seule fois.

Travaux Pratiques

1 Découverte de Python

Exercice 1

L'objectif de ce TP est de vous familiariser avec Python.

`python` peut être lancé de deux manières :

- via la console : tapez `python3` dans un terminal
- en exécutant un fichier : tapez `python3 nom_fichier.py` dans un terminal

1.1. Dans la console, tapez les commandes suivantes :

```
>>> 5+7
>>> 34/9
>>> min(3,5)
```

Les variables s'affectent par le signe `=` :

```
>>> a = 3+4
```

L'affichage d'une variable dans la console se fait juste en tapant son nom ou `print(nom_variable)`

```
>>> a
>>> print(a)
```

En `python`, les deux valeurs binaires sont `True` et `False`.

1.2. (Opérateurs binaires) Exécutez les commandes suivantes :

- `True and False`
- `True or False`
- `not(True) or False and True`

==	teste l'égalité entre deux valeurs
!=	teste la différence entre deux valeurs
<, <=, >, >=	teste les relations d'ordre entre deux valeurs

Comparaisons

Exercice 2 `if`, `while`, `for`, `range` et fonctions

Voici un exemple d'utilisation en python de l'instruction `if` :

```
if x>0 :
        y=x
        print("Cas positif")
elif x<0 :
        y=-x
        print("Cas négatif")
else :
        y=0
        print("Cas nul")
print("Fin")
```

Noter l'absence d'accolades et l'utilisation de l'indentation : les blocs de l'instruction conditionnelle ne sont distinguables que par l'indentation. Il en est de même pour les boucles `for`, `while` et pour l'écriture des fonctions.

Voici un exemple de fonction écrite en python :

```
def f(x) :
        if x >= 0 :
                return x
        else :
                return -x
```

2.1. Dans un fichier `tutoriel.py`, écrivez une fonction g permettant de calculer :

$$g(x) = \begin{cases} -x^2 & \text{si } x < 0 \\ x^2 & \text{sinon} \end{cases}$$

`python` est surtout un langage de script. Si vous souhaitez tester votre fonction, il vous suffit de rajouter, par exemple,

```
print(g(-4))
```

à la fin de votre fichier et de le lancer dans le terminal par la commande :

```
$ python3 tutoriel.py
```

En cours, nous avons vu que que la commande `for` s'utilise de la manière suivante :

```
for i in range(a,b) :
        ...
```

2.2. Afficher ce que donne les commandes :

- ```
  for i in range(5,10) :
          print(i)
  ```
- ```
  for i in range(6) :
          print(i)
  ```
- ```
  for i in range(30,3,-8) :
          print(i)
  ```

2.3. Écrivez une fonction h prenant en paramètre un entier positif x et affichant tous les entiers multiples de 7 compris entre 0 et x inclus.

Remarque : dans la suite des TPs, on n'utilisera pas ***`python`*** *directement dans le terminal. Les fonctions seront à écrire dans des fichiers et seront testées et utilisées à la fin de ces fichiers, qu'on exécutera en entier.*
Exemple :

```
def f(a,b) :
    ...

def g(a) :
    ...

...

print("f(3,\"chat\") =",f(3,"chat"))
...
```
```

2 Instructions de bases, entrées-sorties

Exercice 3 Ou exclusif

Il n'y a pas dans le cours de test "ou exclusif" (ou xor) qui renvoie `VRAI` si et seulement si l'un des deux paramètres est vrai, mais pas les deux simultanément.

3.1. Écrivez une fonction `python` prenant en paramètres 2 booléens et renvoyant le "ou exclusif" de ces deux paramètres.

Exercice 4

4.1. Chacun de ces programmes est faux. Préciser l'erreur.

a.
```
a = 5
if a%0==2 : print ("Nbr pair")
```

b.
```
a == 5
if a : print ("Nbr non nul")
```

Exercice 5

Considérons le petit programme suivant :

```
a = 'False'
if a :
    print(4)
else :
    print(8)
```

5.1. À l'exécution le programme affiche 4. Pourquoi ?

Exercice 6

6.1. Écrire une fonction prenant en paramètre un entier n et qui permet de savoir si n est pair ou non : si l'entier n est pair, l'ordinateur doit afficher "n est pair", sinon l'ordinateur doit afficher "n est impair" (la fonction ne doit rien renvoyer).

Tester ce code lorsque `n` $= 46$ puis lorsque `n` $= 47$.

Exercice 7

7.1. Écrire une fonction prenant en paramètre un entier n, qui affecte à une variable `res` la valeur 1 si n est un multiple de 3 et 0 si n n'est pas un multiple de 3, et qui renvoie `res`.

Tester votre programme avec $n = 15$ puis avec $n = 19$.

Exercice 8 Payer ses impôts

Lorsqu'on ne dépasse pas la tranche à $5,5\%$, le montant de l'impôt que l'on doit payer, en fonction de son revenu pour une personne ne possédant qu'une seule part, est défini par la formule :

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{sinon} \end{cases}$$

8.1. Écrire une fonction `calcul_impot_1` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer.
Tant qu'on ne dépasse pas la tranche à 14%, le montant de l'impôt est

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{si } 5875 \leqslant revenu < 11\,720 \\ \frac{14}{100}revenu - 1\,319.33 & \text{sinon} \end{cases}$$

8.2. Écrire une fonction `calcul_impot_2` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer selon ce nouveau calcul, en utilisant une instructions `elif`.

8.3. Faire des tests dans chacun des trois cas suivants :

- Le revenu vaut 3 000 (l'impôt calculé doit valoir 0)
- Le revenu vaut 7 000 (l'impôt calculé doit valoir 61.875)
- Le revenu vaut 20 000 (l'impôt calculé doit valoir 1 480.67)

8.4. Écrire une fonction `calcul_impot_3` effectuant le même calcul que `calcul_impot_2`, mais sans utiliser d'instruction `elif`.
Refaire les trois tests de la question ci–dessus et vérifier la validité.
Pour un revenu quelconque (toujours avec une seule part), l'impôt est :

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{si } 5875 \leqslant revenu < 11\,720 \\ \frac{14}{100}revenu - 1\,319.33 & \text{si } 11\,720 \leqslant revenu < 26\,030 \\ \frac{30}{100}revenu - 5\,484.13 & \text{si } 26\,030 \leqslant revenu < 69\,783 \\ \frac{40}{100}revenu - 12\,462.43 & \text{au–dessus} \end{cases}$$

8.5. Écrire une fonction `calcul_impot_4` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer selon ce nouveau calcul.
Sur Mars, le calcul de l'impôt dépend du revenu et d'un booléen permettant de savoir si la personne imposée habite dans un cratère ou non.

Le calcul de l'impôt est :

	`cratere`	
	`True`	`False`
`revenu` $< 4\,800$	$0.12 *$ `revenu`	0
`revenu` $\geqslant 4\,800$	$0.25 *$ `revenu` $- 624$	$0.12 *$ `revenu` $- 576$

8.6. Écrire une fonction `calcul_impots_mars` prenant en paramètre un entier `revenu` et un booléen `cratere` et renvoyant le montant de l'impôt.
Effectuer les tests suivants :

- un martien habitant un cratère et touchant un revenu de 4 000 doit payer 480 ;
- un martien habitant un cratère et touchant 6 000 doit payer 876 ;
- un martien n'habitant pas de cratère et touchant 6 000 doit payer 144.

Exercice 9 Importance de l'indentation

On cherche à calculer $\sum_{i=1}^{10} \frac{1}{i}$. Considérons les deux programmes suivants :

```
s = 0
for i in range (1, 11) :
    s += 1/i
print (s)
```

```
s = 0
for i in range (1, 11) :
    s += 1/i
    print (s)
```

9.1. Quelle différence constate–t–on lors de l'exécution entre les deux programmes ?

Exercice 10 Calculs de sommes et de produits

Soient $S(n) = \sum_{k=1}^{n} \frac{1}{k}$ et $H(n) = \prod_{k=2}^{n} \left(1 - \frac{1}{k^2}\right)$ (pour $n \geqslant 2$).

10.1. Nous allons calculer $S(n)$ avec l'algorithme suivant :

Algorithme 1 : somme_S(n : entier)

Données :
k : entier
s : réel

```
1 s ← 0
2 k ← 0
3 tant que k < n faire
4 |  k ← k+1
5 |  s ← s+1/k
6 fin
7 retourner s
```

10.2. Programmer cet algorithme en `python`.

10.3. Test : vérifier que $S(10) \simeq 2.9289$

10.4. Si dans le test de la boucle `while` on avait remplacé le test $k < n$ par $k \leqslant n$, qu'aurait calculé ce programme ?

10.5. Modifier l'algorithme de la question précédente pour calculer le produit $H(n)$.

Test : vérifier que $H(5) = 0.6$.

10.6. Déterminer le plus petit entier n tel que $S(n) \geqslant 10$. On pourra par exemple appliquer l'algorithme suivant :

Données :
n : entier
s : réel

```
1 n ← 0
2 s ← 0
3 tant que s < 10 faire
4 |  n ← n+1
5 |  s ← s+1/n
6 fin
7 retourner n
```

Élement de réponse : on doit trouver un entier compris entre 10 000 et 100 000.

Le savant Sunisoc conjecture que $\lim_{n\to+\infty} H_n = \frac{1}{2}$ (cette conjecture est vraie).

On fixe un réel $\varepsilon > 0$ "petit" et on cherche à déterminer le plus petit entier $n \geqslant 2$ tel que $\frac{1}{2} - \varepsilon \leqslant H_n \leqslant \frac{1}{2} + \varepsilon$.

10.7. Écrire un programme permettant de résoudre automatiquement ce problème.
Test : pour $\varepsilon = 0.002$ on doit trouver $n = 250$.

Exercice 11 Itérés d'une suite récurrente d'ordre 1

On définit une suite u_n par $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = \frac{6+u_n}{6-u_n}$.

Nous allons calculer u_n avec l'algorithme suivant :

Algorithme 2 : calcul_u(n : entier)

```
   Données : u : réel
1  u ← 0
2  tant que n > 0 faire
3  |  n ← n-1
4  |  u ← (6+u)/(6-u)
5  fin
6  retourner (u)
```

11.1. Programmer cet algorithme en `python`.

Test : vérifier que $u_5 \simeq 1.812$.

Si dans le test de la boucle `while` on avait remplacé le test $n > 0$ par $n \geqslant 0$, qu'aurait calculé ce programme ?

Et si on avait mis le test $n < 0$?

11.2. Calculer $\sum_{k=0}^{n} u_k$.

Test : pour $n = 5$ on doit trouver environ 7.5534

Exercice 12 Gestion de matchs

On souhaite gérer les matchs au cours d'une année d'une ligue de sport.

Rappel : usage d'une commande d'impression : Si `i = 3` et `j = 2` sont définis, la commande :

```
print ("L'equipe",i,"se deplace contre l'equipe",j)
```

permet d'afficher le message : `L'equipe 3 se deplace contre l'equipe 2`

12.1. Une équipe doit jouer contre toutes les autres équipes. Écrire une fonction prenant en paramètre un entier `k` représentant le numéro de l'équipe et un entier `n` représentant le nombre total d'équipes et qui permet d'afficher tous les messages :

```
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 2
L'equipe 2 se deplace contre l'equipe 3
...
```

12.2. On cherche maintenant à écrire un message pour chacun des matchs à jouer. Écrire une fonction prenant en paramètre un entier `n` représentant le nombre total d'équipes et qui affiche les messages :

```
L'equipe 1 se deplace contre l'equipe 1
L'equipe 1 se deplace contre l'equipe 2
L'equipe 1 se deplace contre l'equipe 3
...
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 2
L'equipe 2 se deplace contre l'equipe 3
...
L'equipe 3 se deplace contre l'equipe 1
L'equipe 3 se deplace contre l'equipe 2
L'equipe 3 se deplace contre l'equipe 3
...
```

12.3. Dans un vrai tournoi de ligue une équipe ne se déplace pas pour jouer contre elle–même. Rajouter une commande `if` pour n'afficher le message que si les deux numéros d'équipe sont différents. On doit obtenir un résultat du type :

```
L'equipe 1 se deplace contre l'equipe 2
L'equipe 1 se deplace contre l'equipe 3
L'equipe 1 se deplace contre l'equipe 4
...
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 3
L'equipe 2 se deplace contre l'equipe 4
...
L'equipe 3 se deplace contre l'equipe 1
L'equipe 3 se deplace contre l'equipe 2
L'equipe 3 se deplace contre l'equipe 4
...
```

12.4. Pour cause de budget restreint la ligue de sport décide que deux équipes ne joueront pas un match aller et un match retour mais un seul match sur terrain neutre. Les messages à afficher sont maintenant :

```
Matchs de l'equipe 1

Matchs de l'equipe 2
L'equipe 2 joue contre l'equipe 1

Matchs de l'equipe 3
L'equipe 3 joue contre l'equipe 1
L'equipe 3 joue contre l'equipe 2

Matchs de l'equipe 4
L'equipe 4 joue contre l'equipe 1
L'equipe 4 joue contre l'equipe 2
```

```
L'equipe 4 joue contre l'equipe 3
...
```

Pour les plus rapides

Exercice 13

Regardons tous les nombres $1 \leqslant k < 10$ multiples de 3 ou de 5 : il y a 3, 5, 6 et 9. Leur somme vaut 23.

13.1. Calculer la somme de tous les entiers $1 \leqslant k < 1000$ multiples de 3 ou de 5. Réponse : 233 168

Exercice 14 Transmission imparfaite avec code auto–correcteur

Deux dispositifs informatiques échangent des données : l'émetteur envoie cinq entiers a_1, a_2, a_3, $b = a_1 + a_2 + a_3$ et $c = a_1 + 2a_2 + 3a_3$.
À cause d'imperfections techniques, il y a un risque faible mais non négligeable que l'un de ces entiers ait été modifié pendant la transmission de données : le récepteur reçoit des entiers ar_1, ar_2, ar_3, br et cr dont l'un est potentiellement différent de l'entier émis. Le risque que deux entiers (ou plus) aient été mal transmis est en pratique trop faible pour qu'on en tienne compte.

Votre but est, connaissant les entiers ar_1, ar_2, ar_3, br et cr, de déterminer s'il y a eu ou non une erreur de transmission et, si c'est le cas, de retrouver les valeurs des entiers émis.

14.1. Commençons par envisager le cas où l'un des entiers a_1, a_2 ou a_3 a été mal transmis (mais b et c ont été correctement transmis). Montrer que dans ce cas $br \neq ar_1 + ar_2 + ar_3$ et $cr \neq ar_1 + 2ar_2 + 3ar_3$. Expliquer comment déterminer lequel parmi a_1, a_2 ou a_3 a été mal transmis, et comment corriger l'erreur.

14.2. Que se passe–t–il si a_1, a_2 et a_3 ont été correctement transmis, mais qu'il y a eu une erreur de transmission ou bien sur b ou bien sur c ?

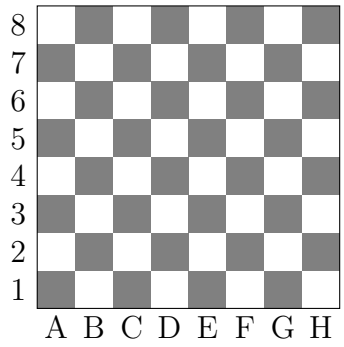
14.3. Écrire une fonction `code_correcteur` prenant en paramètres 5 entiers `ar1`, `ar2`, `ar3`, `br` et `cr`, et qui affiche suivant les cas l'un des messages suivants :

- `Les cinq entiers ont été correctement transmis`
- `a1 a été mal transmis : sa vraie valeur est :` *suivi de la valeur de* a_1
- *(ou un message similaire avec* a_2*,* a_3*,* b *ou* c*)*

Tester votre programme dans chacun des cas suivants :

- $ar1 = 3$, $ar2 = 2$, $ar3 = 6$, $br = 10$, $cr = 25$. On doit obtenir le message :
  `b a été mal transmis, sa vraie valeur est : 11`
- $ar1 = 3$, $ar2 = 1$, $ar3 = 6$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `a2 a été mal transmis, sa vraie valeur est : 2`
- $ar1 = 3$, $ar2 = 2$, $ar3 = 6$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `Les cinq entiers ont été correctement transmis`
- $ar1 = 3$, $ar2 = 2$, $ar3 = 5$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `a3 a été mal transmis, sa vraie valeur est : 6`

Exercice 15 Mouvement du cavalier sur un échiquier



Un échiquier est composé de 8×8 cases, chaque case étant repérée par son abscisse (A, B, C,. . ., H) et par son ordonnée (1, 2, 3,. . ., 8).
Nous allons coder chaque case par une chaîne de 2 caractères : par exemple la valeur `"F4"` désignera une case (ne pas oublier les guillemets).

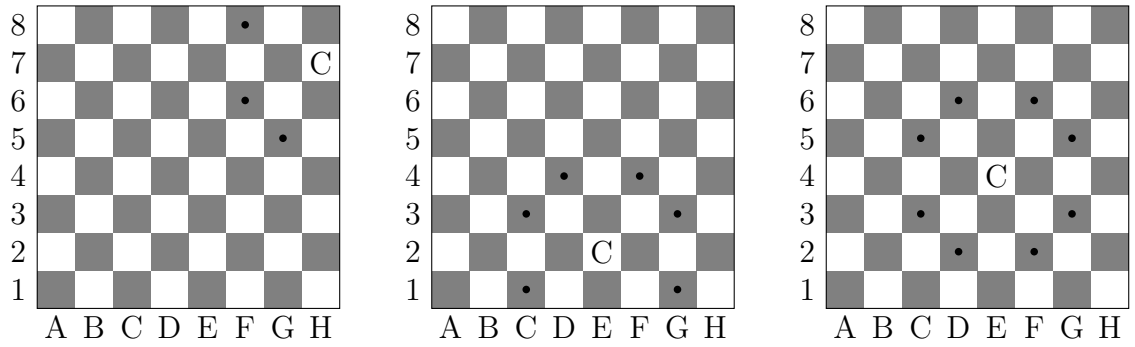
15.1. Tester les instructions suivantes (inutile de recopier les commentaires derrière le symbole `#`) :

```
>>> c1 = "F4"           # definition d'une case c1
>>> x1 = c1[0]          # abscisse de c1 : attention aux crochets
>>> y1 = c1[1]          # ordonnee de c1
>>> type (c1)           # type str (string) : chaine de caracteres
>>> x1
>>> type (x1)
>>> y1
>>> type (y1)           # encore de type str
>>> c2 = "G2"           # definition d'une autre case c2
>>> x2 = c2[0]
>>> y2 = c2[1]
>>> y1 - y2             # attention : erreur
>>> y1 = int (y1)       # convertion en int (integer : entier)
>>> type (y1)           # la convertion a bien eu lieu
>>> y2 = int (y2)
>>> y1 - y2             # maintenant ca marche
>>> x1 - x2             # erreur, bien sur
>>> ord (x1)            # code entier du caractere "F"
>>> ord (x1) - ord (x2) # calcul de la difference des ordonnees
>>> chr (66)            # reciproque de la fonction ord
>>> str (8)             # convertion d'un entier en str
>>> chr(67) + str (8)   # concatenation (mise bout-a-bout)
```

Un cavalier peut se déplacer (en un seul mouvement) d'une case c_1 vers une case c_2 exactement dans chacun des deux cas suivants :

- ou bien leur abscisse diffère de 2 et une ordonnée diffère de 1 ;
- ou bien leur abscisse diffère de 1 et leur ordonnée diffère de 2.

Bien sûr il est nécessaire que les deux cases c_1 et c_2 soient des cases qui existent sur l'échiquier ! Voici ci–dessous trois cavaliers (repérés par la lettre "C") ainsi que, à chaque fois, les cases sur lesquelles il peut se déplacer.



15.2. Écrire une fonction `deplacement_possible` prenant en paramètre deux chaînes de caractères `c1` et `c2` représentant des cases de l'échiquier et qui renvoie `True` si un cavalier peut passer de la case `c1` à la case `c2`. On pourra utiliser la fonction `abs` qui calcule la valeur absolue.

Effectuer les tests suivants :

- Pour les cases `"F4"` et `"G2"` le déplacement est possible ;
- pour les cases `"A3"` et `"B4"` le déplacement est impossible ;
- pour les cases `"G7"` et `"I8"` le déplacement est impossible (la case `"I8"` n'existe pas).

15.3. Écrire une fonction `cases_possible` prenant en paramètre une chaîne de caractères `c1` représentant une case existante et qui affiche les cases possibles accessibles par un cavalier en un seul mouvement depuis `c1`.

Tester votre programme de déplacement du cavalier :

- Depuis la case `"H1"` votre programme doit afficher les cases `"G3"` et `"F2"` ;
- depuis la case `"D2"` votre programme doit afficher les cases `"B1"`, `"B3"`, `"C4"`, `"E4"`, `"F3"` et `"F1"`.

Exercice 16

Soient $k, n \in \mathbb{N}$ tels que $k \leqslant n$.

Le coefficient binomial $\binom{n}{k}$ peut se calculer avec la récurrence :

$$\binom{n}{0} = 1 \text{ et } \binom{n+1}{k+1} = \frac{n+1}{k+1}\binom{n}{k}$$

16.1. Écrire une fonction `binomial` prenant en paramètre deux entiers positif `n` et `k` et retournant la valeur de $\binom{\texttt{n}}{\texttt{k}}$.

On ne demande pas de prouver cette formule, mais de l'appliquer pour calculer $\binom{15}{10}$ *(réponse : 3003).*

Exercice 17

Les nombres triangulaires T_n, pentagonaux P_n et hexagonaux H_n sont définis ainsi :

Triangulaires : $T_n = \dfrac{n(n+1)}{2}$ (on trouve $1, 3, 6, 10, 15 \dots$)

Pentagonaux : $P_n = \dfrac{n(3n-1)}{2}$ (on trouve $1, 5, 12, 22, 35, \dots$)

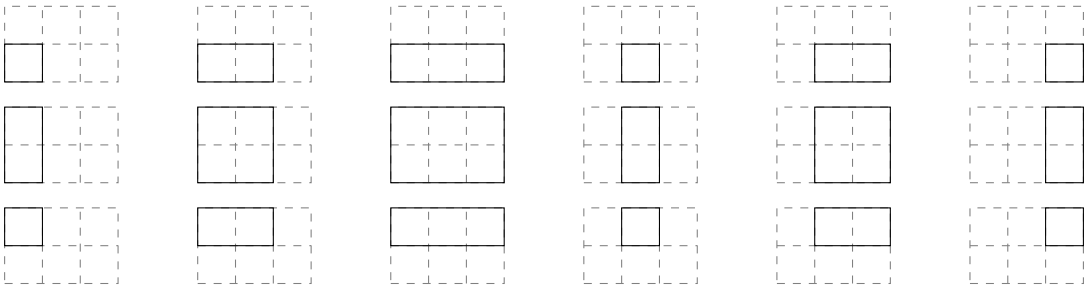
Hexagonaux : $H_n = n(2n-1)$ (on trouve $1, 6, 15, 28, 45, \ldots$)

Il est possible de vérifier que $T_{285} = P_{165} = H_{143} = 40755$.

17.1. Trouver le nombre suivant qui est à la fois triangulaire, pentagonal et hexagonal (il est supérieur à un milliard...)

Exercice 18 Dénombrer les rectangles

Constatons que sur une grille de taille 3×2 on peut positionner 18 rectangles :



Il n'est pas possible de trouver de grille contenant exactement 2 000 000 (2 millions) de rectangles.

18.1. Calculer cependant les dimensions d'une grille qui s'en approche le plus.