Introduction à la Programmation en Python

Cours et Travaux Dirigés — Première Partie

Licence
Université de Paris
Année académique 2020–2021

Introduction à la Programmation 1 PYTHON

MT11Y060

Séance 1 de cours/TD

Université de Paris

Objectifs:

- Utiliser PYTHON comme une calculatrice.
- Identifier et donner un sens aux différentes constructions du langage (affectation et utilisation des variables, boucles for, conditionnelles, procédures et fonctions).
- Apporter une modification mineure à un programme existant.

1 De quoi est fait un programme ?

Qu'est-ce qu'un programme ? [COURS]

- Un **programme** est la représentation d'une séquence d'instructions à exécuter. Par exemple, une recette est un programme exécuté par un cuisinier. Un code source PYTHON est un programme exécuté par un ordinateur. En IP1, nous utiliserons la version 3 de PYTHON.

Qu'est-ce qu'un ordinateur ? [COURS]

- En première approximation, un **ordinateur** est une machine à calculer munie d'une mémoire et connectée à des périphériques (comme un écran, une carte réseau, etc).
- Un ordinateur peut soit **faire des calculs** qui produisent des **valeurs** ; soit **effectuer des actions** en transmettant des valeurs à sa mémoire ou à des périphériques.
- Les calculs peuvent dépendre de valeurs récoltées par les périphériques ou lues dans la mémoire.

Les constituants d'un programme [COURS]

- Les **expressions** décrivent des calculs à faire pour produire des valeurs.
- Les **instructions** décrivent des actions à effectuer.
- Parmi les instructions il y a les **affectations**, elles stockent des données dans des **variables**.

```
    startSecs = (startHour * 60 + startMin) * 60
    numberOfSecs = stopSecs - startSecs
    alertCode = 0
    for i in range (0, numberOfSecs, 1):
        time.sleep(1)
        if (numberOfSecs - i < 30):
            alertCode = 1
        printInt(numberOfSecs - i, alertCode)
```

Listing 1 – Que fait cette séquence d'instructions?

Exercice 1 (Premier décodage, ⋆)
Dans le programme PYTHON *précédent, classifiez les parties du code source correspondant aux expressions, aux instructions et aux affectations.* □

2 Le langage des expressions arithmétiques

Sous-langage des expressions arithmétiques [COURS]

- Les expressions sont classifiées à l'aide de **types** correspondant à la forme des valeurs qu'elles calculent.
- Pour connaître le type d'une expression `exp`, on peut taper dans l'interpréteur PYTHON la commande `type(exp)` où `exp` est une expression à calculer.
- Le type `int` est celui des expressions qui calculent des valeurs entières, les expressions arithmétiques.
- Le type `int` contient tous les entiers représentables dans la limite de la mémoire de l'ordinateur.
- Une **expression arithmétique de type** `int` peut être :
  (a) une constante entière (`0`, `1`, `2`, `-1`, `-2`, `-2147483648`, `21474836472133929101012311103` ...) ;
  (b) une variable, par exemple `x`, dont la valeur doit être entière ;
  (c) deux expressions séparées par une opération arithmétique binaire (`1 + 2`, `1 + 2 * 3 // 4`, ...) ;
  (d) une expression entourée de parenthèses (`(1 + 2)`, `(1 + 2 * 3 // 4)`, ...) ;
  (e) une expression précédée du signe moins (`-2`, `-(1 + 2)`, ...).
- Il y a une autre construction importante qui est autorisée dans les expressions : les appels à des fonctions qu'on verra un peu plus tard dans ce cours.
- Les opérateurs ont des priorités relatives, par exemple
  {`*`, `//`, `%`} $\preceq$ {`+`,`-`}
  où $\preceq$ signifie "être prioritaire sur".
- Les opérateurs binaires `+`,`-`,`*`,`/`,`%`,`//` sont associatifs à gauche.
- On peut toujours rajouter des parenthèses pour expliciter la priorité de certains calculs sur d'autres.
- L'évaluation de l'expression "`1 + 2 * 3 - 4`" est équivalente à l'expression "`1 + (2 * 3) - 4`" et elle se décompose en (i) "`2 * 3`" donne 6 ; (ii) "`1 + 6`" en 7 et (iii) "`7 - 4`" donne 3.
- Les opérations ($+$, $-$, $//$, $*$, ...) ont le sens usuel *tant que l'on reste entre le cadre du type* `int`.
- Dans le type `int`, la division $//$ est une *division entière*. Par exemple, "`31 // 7`" vaut 4.
- L'opération $/$ entre deux valeurs de type `int` ne renvoie pas nécessairement une valeur de type `int` mais de type `float` qui correspond aux réels.
- L'opérateur `%` désigne le *modulo* : "`a % b`" est le reste dans la division entière de `a` par `b`.
  Par exemple, "`31 % 7`" vaut 3 car $31 = 7 \times 4 + 3$.

Exercice 2 (Python comme une calculatrice, ⋆)
Prévoir l'évaluation des expressions arithmétiques suivantes :

```
6 * 7 + 3
6 * (7 + 3)
45 // 7
3 * 7 // 4
(3 * 7) // 4
(45 // 7) * 7 + 45 % 7
(1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12 - 13) // (1 - 2 + 3 -
    4 + 5 - 6 + 7 - 8 + 9 - 10 + 11 - 12 + 13)
```

□

3 Utilisation des variables

Utilisation des variables [Cours]

— Une **variable** a un *nom* et contient une *valeur*, c'est-à-dire le résultat d'un calcul.
— En Python, une variable est créée par une affectation à celle-ci.
— On met dans la variable `x` de type `int` le résultat du calcul `6 * 7` qui est de type `int` ainsi :

```
1 x = 6 * 7
```

— Dans l'exemple précédent, la valeur de la variable `x` est 42.
— Exemples de noms de variable : `x`, `y`, `foo`, `foobar42`...
— On peut utiliser la valeur d'une variable dans une expression en faisant référence à son nom. Ainsi, l'expression "`x + 1`" vaut 43 si `x` vaut 42.
— On peut changer la valeur d'une variable qui a été affectée auparavant en lui *affectant* le résultat d'un nouveau calcul. L'opérateur d'affectation est « = » :

```
1 x = 2 * 10
```

— Sur papier, une mémoire contenant les variables `x = 42`, `y = 73` et `z = 37` sera écrite :

x	y	z
42	73	37

— Quand on essaye d'utiliser dans une expression une variable qui n'a pas encore été créée il y a évidement une erreur.

Instructions [Cours]

— Une expression calcule une valeur tandis qu'une instruction a un effet sur la machine.
— Une affectation est un exemple d'instruction.
— On dit qu'une machine exécute une instruction.

Exercice 3 (Nommer, ⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 x = 6 * 7
2 y = x + x
```

□

Exercice 4 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 x = 1
2 y = 4
3 x = y + 2
```

□

Exercice 5 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 x = 1
2 y = x - 1
3 x = 2
```

□

Exercice 6 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
x = 1
x = x - 1
```

□

4 Fonctions et procédures

Nommer un calcul comme une fonction [Cours]

— Lorsqu'un calcul doit être réalisé plusieurs fois par un programme avec seulement des paramètres différents, on peut l'isoler en définissant une fonction.
— La fonction qui attend en paramètre une valeur `x` et calcule une valeur qui est le triple de `x` s'écrit :

```
def triple (x) :
    return (x * 3)
```

— Dans cet exemple, on remarque un espace avant la deuxième ligne, il s'agit d'un espace d'**indentation** et il est nécessaire pour indiquer que cette ligne appartient à la fonction. Cette ligne définie le **corps de la fonction**.
— Les espaces d'indentation peuvent s'écrire à l'aide du caractère de tabulation ou du caractère espace.
— Même si en Python, on ne précise pas le type des données manipulées, il est important d'avoir cette information toujours en tête. Par exemple, la fonction `triple` peut prendre en entrée une donnée de type `int` et elle renverra une donnée de type `int`.
— On peut utiliser une fonction en écrivant son nom suivi de son paramètre entouré de parenthèses, comme par exemple dans l'expression :

```
triple (2) * 6
```

— Dans cet exemple, `triple(2)` vaut 6 donc l'expression vaut 36.
— Voici des exemples de noms de fonction : `triple`, `add3`, `fooBar`, ...
— Une fonction peut attendre plusieurs paramètres qu'on les sépare par des virgules :

```
def multiplication (x, y) :
    return (x * y)
```

— Le corps d'une fonction peut avoir plusieurs lignes :

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

— On peut utiliser des fonctions prédéfinies dans des *bibliothèques de fonctions*, par exemple la fonction `abs` est prédéfinie dans la bibliothèque standard de Python.

Exercice 7 (Utiliser une fonction, ⋆)
Étant donnée la fonction suivante, que vaut l'expression `twice(3)` *?*

```
def twice (x) :
    return (2 * x)
```

□

Exercice 8 (Utiliser une fonction, ⋆⋆)
On considère la fonction suivante.

```
def cube (x) :
    return (x * x * x)
```

Quelle est la valeur de la variable `a` *à la suite des instructions suivantes ?*

```
b = 5
a = 3
a = b - a
a = cube(a)
```

□

Exercice 9 (Écrire une fonction, ⋆⋆)
Voici une fonction

```
def f (x) :
    return (x * x + 5)
```

Quel est son nom ? Que calcule-t-elle ? Modifiez son corps pour qu'elle calcule la division entière par 3. Modifiez son nom pour qu'elle se nomme `third`.

□

Exercice 10 (Écrire une fonction, ⋆⋆)
Écrire des fonctions effectuant les calculs suivants :

1. *La puissance 5 d'un entier donné en paramètre.*
2. *Le produit de deux entiers donnés en paramètre moins leur somme.*
3. *Le produit de trois entiers donnés en paramètre au carré.*

□

Nommer une suite d'instructions comme une procédure [Cours]

— Une fonction est supposée renvoyer un résultat d'un calcul par le bias de l'instruction `return`. Lorsque nous voulons répéter une suite d'instructions qui affectent l'état de la machine sans renvoyer un résultat en particulier, nous pouvons définir des **procédures**.

— Un **appel de procédure** est une instruction écrite à l'aide du nom de la procédure suivi d'un ou plusieurs paramètres séparés par des virgules et entourés de parenthèses. Par exemple :

```
print (1 + 2)
```

affiche 3 à l'écran à l'aide de la procédure prédéfinie `print`. Un autre exemple :

```
putPixel (0, 0, 255, 255, 255)
```

affiche un pixel blanc en position (0, 0) d'une image à l'aide de la procédure prédéfinie `putPixel`.

— Une procédure est définie en termes d'une suite d'instructions.

— Par exemple, la procédure qui attend deux entiers `x` et `y` et qui affiche successivement leur somme et leur produit s'écrit :

```
def showProductAndSum (x, y) :
    print (x + y)
    print (x * y)
```

— On remarquera que contrairement aux fonctions, les procédures ne renvoient pas de valeurs !

Exercice 11 (Différences syntaxiques, $\star\star$)

1. *Quelles différences trouvez-vous entre la syntaxe de déclaration des fonctions et des procédures ?*
2. *Même question pour l'appel de fonctions et l'appel de procédures.*

□

5 Conditionnelle

Instruction conditionnelle [Cours]

— Une instruction conditionnelle permet d'exécuter des instructions en fonction d'une condition.
— On écrit par exemple

```
x = 10
y = 42
if (x <= 0) :
    y = x
else :
    y = -x
```

pour affecter la valeur de `x` à `y` si `x <= 0` ou pour lui affecter la valeur `-x` dans le cas contraire.
— Les conditions peuvent par exemple être des comparaisons entre deux expressions à valeur `int` : `e1 == e2`, `e1 != e2`, `e1 < e2`, `e1 <= e2`, `e1 > e2`, `e1 >= e2`.
— Comme pour les définitions de fonction, il faut faire attention aux espaces d'indentation.

Exercice 12 (Le max, ⋆)
À la suite des instructions ci-dessous, quelle est la valeur de la variable `xy_max` *?*

```
x = 3
y = 4
xy_max = 0
if (x > y) :
    xy_max = x
else :
    xy_max = y
```

Transformez la suite d'instructions pour calculer le minimum de `x` *et* `y` *et le mettre dans une variable* `xy_min`*.*

□

Exercice 13 (Différents tests, ⋆⋆)
Quelle est la valeur des variables `a`, `b` *et* `c` *après la suite d'instructions suivante :*

```
a = 2
b = a * a + 3
c = b - a
if (c == a) :
    a = 1
else :
    a = a + 3
if (b + c < a) :
    b = 2
else :
    b = 4
if (b != c * c) :
    c = 12
else :
    c = -6
```

□

6 Boucles

Boucles bornées [Cours]

— Une boucle bornée permet de répéter plusieurs fois les mêmes instructions.
— Le numéro de l'itération est disponible dans une variable qui s'appelle le *compteur de boucle*.
— La boucle suivante affiche les entiers de 0 à 9 :

```
for i in range (0, 10, 1) :
    print(i)
```

— L'*en-tête de la boucle* est formé du mot-clé `for` suivi de la variable qui prendra une valeur dans la séquence définie par `range (0, 10, 1)`, c'est le compteur de la boucle. La première valeur de cette séquence est 0 ; à chaque coup, on augmente la valeur de `i` de 1 et quand la valeur de `i` vaut 10 alors on "sort" de la boucle.
— Ainsi, l'expression `range (start, stop, step)` définit une séquence de valeurs où :
  — `start` est la valeur initiale de la séquence ;
  — `stop` est la valeur de sortie de la boucle ;
  — `step` est la différence entre deux valeurs successives de la séquence.
— Le *corps de la boucle* vient après l'en-tête de la boucle. Toutes ses lignes commencent par un espace d'indentation et elles sont exécutées à chaque itération de la boucle.
— Nous verrons dans quelque semaines qu'il existe un autre type de boucle introduite par le mot-clé `while`.

Exercice 14 (Afficher des suites d'entiers, ⋆)

1. *Écrivez une boucle qui affiche les 100 premiers entiers, en commençant à 0.*
   Quel est le dernier entier affiché ?
2. *Écrivez une boucle qui affiche les 50 premiers entiers pairs, en commençant à 0.*
   Quel est le dernier entier affiché ?
3. *Écrivez une boucle qui affiche 1000 fois le nombre 3.*

□

7 Do it yourself

Exercice 15 (Valeurs d'expressions entières, ⋆)
Donnez la valeur des trois expressions suivantes : `3 + 5 // 3` `4 * 1 // 4` `2 // 3 * 3 - 2` □

Exercice 16 (Modulo 9, ⋆⋆)
Donnez la valeur des expressions suivantes :
`18 % 9` `81 + 18 % 9` `(81 + 18) % 9`

□

Exercice 17 (Valeur absolue, ⋆)
En vous inspirant de l'exercice 12, donner une suite d'instructions permettant de calculer la valeur absolue d'une variable. □

Exercice 18 (Utiliser une fonction, ⋆)
On considère la fonction suivante.

```
def sumsquare (x, y ) :
    return (x * x + y * y)
```

Quelle est la valeur des variables `a`, `b` *et* `c` *à la suite des instructions suivantes :*

```
a = sumsquare (2, 3)
b = sumsquare (3, 1)
c = sumsquare (4, 3)
```

□

Exercice 19 (Boucles simples, ⋆⋆)

1. *Quel est l'affichage produit par la suite d'instructions suivante :*

```
for i in range (0, 5, 1) :
    print (8)
```

2. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
x = 0
for i in range (0,  5, 1) :
    x = x + 8
```

3. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
x = 0
for i in range (0,  5, 1) :
    x = 10 * x + 8
```

□

Exercice 20 (Des entiers qui s'ajoutent, ⋆⋆)
Quelle est la valeur de `sum` *après la suite d'instructions suivante :*

```
i = 0
sum = 0
sum = sum + i
i = i + 1
sum = sum + i
i = i + 1
sum = sum + i
i = i + 1
sum = sum + i
i = i + 1
sum = sum + i
i = i + 1
sum = sum + i
i = i + 1
```

Si on veut adapter le code ci-dessus pour aller non plus jusqu'à 5 mais jusqu'à 100, 1000 ou plus, on a un problème : on obtiendrait un programme beaucoup trop long !

On peut remplacer la longue liste d'instructions par une boucle `for` *en remarquant que l'instruction "*`sum = sum + i`*" est exécutée 6 fois avec* `i` *prenant pour valeurs les différents entiers entre 0 et 5 car la variable est systématiquement incrémentée (sa valeur est augmentée de 1) grâce à l'instruction "*`i = i + 1`*".*

On obtient ainsi la boucle :

```
bound = 6
sum = 0
for i in range (0, bound, 1) :
```

```
4      sum = sum + i
```

1. *On remplace la première ligne du code précédent par "*`bound = 100`*". Quelle est la valeur de* `sum` *après l'exécution de cette suite d'instructions ? Même question si on remplace la première ligne par "*`bound = 1000`*" ?*
2. *Modifiez le code précédent pour calculer la somme des entiers de* 10 *à* 100*.*
3. *Écrivez une fonction "*`sumIntegers ( n )`*" qui renvoie la somme des entiers de* 0 *à* n*.*

□

Introduction à la Programmation 1 PYTHON

MT11Y060

Séance 2 de cours/TD

Université de Paris

Objectifs:

- Découverte du type `bool` et `str`.
- Comprendre qu'il y a des types différents.
- Maîtriser les expressions booléennes dans les conditions.
- Comprendre l'utilisation des boucles dont chaque itération dépend de l'indice.
- Difference entre afficher et renvoyer une chaine de caractères.
- Manipulation des chaines de caractères.

1 Les expressions booléennes : le type `bool`

Des expressions booléennes plus complexes dans les tests [COURS]

- Une expression booléenne a le type `bool` et peut s'évaluer en l'une des deux valeurs `True` ou `False`, ce qui permet l'alternative entre deux actions possibles dans les instructions conditionnelles.
- Une expression booléenne a le type `bool` et peut être construite par exemple à partir des comparateurs, comme `==`, `!=`, `>`, `>=`, `<`, `<=`, et des connecteurs logiques, comme `and`, `or`, `not`. Exemples d'expressions booléennes :
  - `5 != 4`
  - `2 < x and  x <= 5`
  - `x == 1 or x == 2`
  - `not (x == 0)` (qui est équivalent à "`x != 0`")
- Attention à ne pas confondre `=` et `==` : Le premier fait partie d'une instruction d'affectation, tandis que le deuxième est un opérateur de comparaison et est donc à utiliser dans une expression booléenne.
- Une expression avec un **et** (en PYTHON `and`) est vraie si les expressions à gauche et à droite du **et** sont vraies.
- Une expression avec un **ou** (en PYTHON `or`) est vraie si au moins une des expressions à gauche et à droite du **ou** est vraie.
- Une expression avec une négation (en PYTHON `not`) est vraie si l'expression après la négation est fausse.
- L'opérateur unaire `not` est prioritaire sur l'opérateur `and` qui est lui-même prioritaire sur l'opérateur `or`.
- On peut combiner ces opérateurs, par exemple en écrivant `(x == 1 or  x == 2) and y > 5`
- Chacune de ces expressions booléennes a donc une valeur qui est soit `True` soit `False`.

Exercice 1 (Savoir évaluer une condition, ⋆)
Quelle est la valeur de la variable `x` *après la suite d'instructions suivante ?*

```
a = 2
a = a * a * a * a + 1
```

```
b = a // 2
c = a - 1
x = 0
if ((b == 0) and (c == 16)) :
    x = 1
else :
    x = 2
```

□

Exercice 2 (Équivalence d'expressions, ⋆⋆)
On dit que deux expressions booléennes sont équivalentes *quand ces deux expressions s'évaluent toujours vers la même valeur booléenne pour toutes les valeurs possibles des variables qui apparaissent dans ces expressions.*

Par exemple, "`x > 10 and x < 12`*" est équivalente à "*`x == 11`*". Par contre, "*`x > y`*" et "*`x != y`*" ne sont pas équivalentes.*

Dire si les expressions suivantes sont équivalentes (en supposant que `x`*,* `y` *et* `z` *sont des nombres entiers) :*

1. *"*`x > y or x < y`*" et "*`x != y`*";*
2. *"*`x != 3 and x != 4 and x != 5`*" et "*`x <= 2 or x >= 6`*";*
3. *"*`x == y and x == z`*" et "*`x == z`*";*
4. *"*`x == y and x == z`*" et "*`x == y and y == z`*".*

□

Exercice 3 (Simplification des expressions booléennes, ⋆⋆)

Simplifier *une expression booléenne consiste à la remplacer par une autre expression booléenne équivalente qui est plus courte. Simplifier les expressions suivantes :*

- `x > 5 and x > 7`
- `x == y and x == z and y == z`
- `x == 17 or (x != 17 and x == 42)`
- `x > 5 or (x <= 5 and y > 5)`

□

2 Instructions conditionnelles imbriquées

Conditionnelles sans branche négative [Cours]

- La branche négative (qui est exécutée quand la condition est fausse) est *optionnelle*, c'est-à-dire on a le droit de ne pas l'écrire.
- C'est utile s'il n'y a rien à faire quand la condition est fausse. Exemple

```
    if (x <= 0) :
      print("Erreur : x devrait être positif")

```

- La branche positive, par contre, est obligatoire dans une conditionnelle.

Instructions imbriquées [Cours]

— Certaines instructions sont *imbricables* : elles peuvent contenir d'autres instructions.
— Les instructions conditionnelles et les boucles sont des instructions imbricables.
— Voici une instruction conditionnelle imbriquée dans une autre instruction conditionnelle :

```
x = 1
y = 2
z = 0
if (x == 1) :
    if (y == 2) :
        z = 3
    else :
        z = 5
else :
    z = 7
```

— On a le droit d'imbriquer des instructions dans d'autres instructions à volonté : des conditionnelles dans des boucles dans des conditionnelles, etc ...
— Avec des instructions imbriquées sur plusieurs niveaux, il est absolument indispensable de suivre strictement la discipline d'indentation afin de déterminer quelles sont les instructions qui dépendent des conditions précédentes.
— On parlera des boucles imbriquées lors de la séance 3.

Exercice 4 (Comprendre l'imbrication d'instructions, ⋆)
Expliquez la différence du comportement des trois procédures suivantes. Dire pour chacune des trois procédures, pour quelle valeur de `x` *le message "au revoir" est affiché :*

```
def proca(x):
    if (x>0) :
        print("x positif")
        if (x<100) :
            print("x plus petit que 100")
            print("au revoir")

def procb(x):
    if (x>0) :
        print("x positif")
        if (x<100) :
            print("x plus petit que 100")
        print("au revoir")

def procc(x):
    if (x>0) :
        print("x positif")
        if (x<100) :
            print("x plus petit que 100")
    print("au revoir")
```

code/exoIncrementation.py

□

Raccourci pour les conditionnelles en cascade [Cours]

— Lorsqu'un `else` est suivi d'un autre test `if`, on peut utiliser la concaténation `elif` ce qui permet de limiter l'indentation et d'énumérer facilement des cas. Par exemple :

```
x = 3
if (x == 0) :
```

```
  print ("x est nul ")
else :
  if (x == 1) :
    print ("x vaut un ")
  else :
    if (x == 2) :
      print ("x vaut deux")
    else :
      print ("x est négatif ou plus grand que deux")
```

peut s'écrire :

```
x = 3
if (x == 0) :
  print ("x est nul ")
elif (x == 1) :
  print ("x vaut un ")
elif (x == 2) :
  print ("x vaut deux")
else :
  print ("x est négatif ou plus grand que deux")
```

— Dans le cas d'utilisation de tests imbriqués, le premier test validé est pris en compte. Par exemple :

```
x = 2
if (x >= 2) :
  print ("La variable est supérieure à 2")
elif (x <= 3) :
  print ("La variable est inférieure à 3")
else :
  print ("Autre cas")
```

Ce code affiche « `La variable est supérieure à 2` » car le premier test est validé.

Exercice 5 (Utilisation des opérateurs booléens, ⋆⋆)

Réécrire les suites d'instructions suivantes en utilisant moins de tests `if` *:*

1.

```
def f (y) :
    x = 0
    if (y != 0) :
      if (y  <= 10) :
        x = 1
      else :
        x = 2
    else :
        x = 2
    return x
```

Quelle est la valeur finale de `x` *pour chaque valeur de* `y` *prise dans* $\{0, 5, 15\}$ *? Vérifiez que votre réécriture du code est correcte.*

2.

```
def g (a) :
    b = 2 * a
    b = a - a
    if (b == a) :
```

```
        b = 0
    else :
        if (b > 43) :
            b = 0
        else :
            b = 1
    return b
```

Quelle est la valeur finale de `b` *pour chaque valeur de* `a` *prise dans* $\{0, 25, 50\}$ *? Vérifiez que votre réécriture du code est correcte.*

3.

```
def h (c, d) :
    e = 0
    if (d == 6) :
        e = 3
    else :
        if (c >= 2) :
            e = 2
        else :
            e = 3
    return e
```

Quelle est la valeur finale de `e` *pour chaque valeur de* `(c,d)` *prise dans* $\{(0,0),(0,6),(6,0),(6,6)\}$ *? Vérifiez que votre réécriture du code est correcte.*

□

Boucles et conditionnelles [Cours]

Le corps d'une boucle peut contenir n'importe quelle instruction. En particulier, il peut contenir une conditionnelle. Par exemple, le programme suivant affiche `"Pair."` pour les valeurs de `i` qui sont des entiers pairs et `"Impair."` pour les valeurs de `i` qui sont des entiers impairs.

```
for i in range (0, 100, 1) :
    if (i % 2 == 0) :
        print ("Pair.")
    else :
        print ("Impair.")
```

Dans le programme précédent les instructions qui sont effectuées à chaque itération *dépendent de la valeur du compteur de boucle* `i`. On ne se contente donc pas de répéter toujours la même chose : ce que l'on fait à chaque itération dépend du nombre d'itérations déjà effectuées, c'est-à-dire de l'indice de l'itération courante, représenté par la valeur du compteur `i`.

Exercice 6 (Multiples de cinq, ⋆)
Écrire une procédure qui affiche `"Multiple de cinq."` *pour les valeurs de* `i` *entre 0 et 99 qui sont des multiples de cinq et* `"Pas multiple de cinq."` *sinon.* □

Exercice 7 (Multiples de cinq et de deux, ⋆⋆)
Modifier la procédure de l'exercice précédent pour qu'elle affiche `"Pair multiple de cinq."` *pour les valeurs de* `i` *entre 0 et 99 qui sont des entiers pairs et multiples de cinq,* `"Pair pas multiple de cinq."` *pour celles qui sont des entiers pairs qui ne sont pas multiples de cinq, et qui affiche tout simplement* `"Impair."` *pour tous les entiers impairs.* □

3 Les chaînes de caractères : le type `str`

Les chaînes de caractères [Cours]

- Les chaînes de caractères représentent du texte.
- Les constantes de chaînes de caractères sont écrites entre guillemets doubles.
- `"Mr Robot"` est un exemple de chaîne.
- Pour introduire un guillemet dans une chaîne, on écrit `\"`.
- La chaîne vide s'écrit `""`.
- L'opérateur de concaténation de deux chaînes est +.
- Dans une chaîne, la séquence `\n` représente le passage à une nouvelle ligne.
- Les guillemets simples introduisent aussi les chaînes de caractères comme dans `'chaine'`.
- La longueur d'une chaîne de caractères s'obtient grâce à la fonction `len`.
- Ainsi la longueur de la chaîne `"Hello!"` est 6 donc `len("Hello!")` renverra la valeur entière 6.
- L'unique chaîne de caractères de longueur 0 est la chaîne vide `""`.
- **Remarque :** on peut aussi utiliser les comparateurs sur les chaînes de caractères. Ainsi == permet de tester si deux chaînes de caractères sont égales. Et l'ordre > sur les chaînes de caractères est l'ordre lexicographique (celui du dictionnaire) sur les chaînes de caractères. Par exemple, l'expression `"abb" >="aaaaa"` est évaluée à `True`.
- Voici un exemple :

```
def profiler (name) :
    if ((name == "Paul") or (name == "Pierre")) :
        print ("On a trouve Paul ou Pierre.")
    else :
        print ("Ce n'est ni Paul, ni Pierre.")
```

Exercice 8 (De premiers exemples, ⋆)
Quelle est la valeur des variables après l'exécution des instructions suivantes ?

```
s = " rentre chez lui."
s2 = "Paul"
s3 = "Michel"
s4 = s2 + s
s5 = s3 + s
s6 = s2 + " et " + s3 + " rentrent chez eux.\n"
```

□

Différence entre affichage et calcul [Cours]

- Il ne faut pas confondre l'affichage d'une chaîne de caractères sur le terminal et le calcul d'une chaîne.

```
print ("Omar")
```

affiche la ligne `Omar` sur le terminal tandis que le programme suivant n'affiche rien :

```
a = "Omar" + "Roma"
```

mais initialise une variable a avec le résultat du calcul qui concatène les deux chaînes `"Omar"` et `"Roma"` en la chaîne `"OmarRoma"`.

4 Utilisation des différents types `int` et `str`

Les chaînes de caractères et les entiers sont des types différents [Cours]

— Si on met une donnée de type `int` dans une variable `x`, une bonne pratique est d'éviter d'utiliser par la suite cette variable pour stocker une donnée d'un autre type, comme par exemple le type chaîne de caractères.
— Les opérations ne sont pas interprétées de la même façon selon les types. Par exemple on ne peut pas écrire `"a"+3`. Cela produira une erreur à l'exécution du programme qui nous dira que l'opération de concaténation n'est pas possible car la donnée 3 est de type `int`. Il s'agit d'une **erreur de typage**.
— Ainsi la fonction suivante :

```
def add (x) :
    return (x + 2)
```

ne peut pas être appelée avec pour paramètre une donnée de type `str`.
— Certaines fonctions ou procédures acceptent les deux types de données, c'est le cas par exemple de la procédure `print`. Si l'on fait `print ("Hello World!")` la chaîne de caractères `Hello World!` est affichée à l'écran et si l'on fait `print (3)`, le chiffre 3 est affiché à l'écran.
— On peut convertir une donnée de type `int` en chaîne de caractères grâce à fonction `str`. Par exemple, `str(3)` renvoie la donnée `"3"`, c'est-à-dire la chaîne de caractères contenant le caractère 3.
— On peut convertir une chaîne de caractères en une donnée de type `int` grâce à la fonction `int`. Par exemple, `int("3")` renvoie la donnée 3, c'est-à-dire l'entier 3. Attention au cas où la chaîne ne contient pas de caractères numériques, par exemple `int("ab")` génère une erreur.

Exercice 9 (Des exemples simples avec des données de différents type, ⋆)

1. *Quelle est la valeur de la variable* `ch` *après l'exécution des instructions suivantes ?*

```
x = 3
ch = "Salut"
if (x <= 2) :
    ch = "Hallo"
else :
    ch = "Hi"
```

2. *Que fait la suite d'instructions suivantes ?*

```
x = 3
st = "Ca va ?"
ch = ""
x = x // 2
if (x == 1) :
    ch = "How are you ?"
else :
    ch = "Comment allez-vous ?"
ch = ch + "\n"
print (ch)
```

3. *Que fait la suite d'instructions suivantes ?*

```
s = "Anticonstitutionnellement"
a = len (s)
if (a % 2 == 0) :
  print ("La longueur est paire")
else :
```

```
    print ("La longueur est impaire")
```

□

Exercice 10 (Deux représentations très différentes de la même chose, ⋆⋆)

— *Que vaut l'expression* `"41" + "1"` *?*
— *Que vaut l'expression* `41 + 1` *?*
— *Quelle différence faites-vous donc entre la chaîne de caractères* `"42"` *et l'entier* 42 *? Dans quelles situations utiliser l'une ou l'autre de ces représentations de l'entier* 42 *?*

□

Exercice 11 (Trouver le bon type, ⋆⋆⋆)
Pour chacune des fonctions suivantes, précisez le type des données autorisées pour les paramètres et le type de données des valeurs retournées. On ne considérera comme type ici que `int` *et* `str`*.*

```
def f (x) :
    y = x % 2
    if (y == 0) :
        return "Argument pair"
    else :
        return "Argument impair"

def g (x) :
    y = x % 2
    if (y == 0) :
        print ("Argument pair")
    else :
        print ("Argument impair")
    return y

def h (x, y) :
    return (x + y)

def id (x) :
    return x
```

code/exoInference.py

□

5 Fonctions manipulant le type `str`

Les fonctions peuvent utiliser le type `str` [Cours]

— Une fonction peut renvoyer une valeur de type `str`.
— Les valeurs de type `str` peuvent être utilisées comme paramètres d'une fonction.

On considère les deux fonctions données ci-dessous :

```
def sandwich (s, s2) :
    return s + s2 + s

def neymarOrNot (x) :
    st = ""
    if (x == 10) :
        st = "C'est le numéro de Neymar."
```

```
    else :
        st = "Ce n'est pas le numéro de Neymar."
    return st
```

Elles peuvent être utilisées de la façon suivante :

```
s = sandwich ("Hello", "World!\n")
s2 = sandwich (s, "Ca va ?")
s3 = neymarOrNot (1)
```

Voici quelques fonctions et procédures pour manipuler des chaînes de caractères :
- Comme dit précédemment, la fonction `len` permet de calculer la longueur d'une chaîne de caractères.
- La fonction `str` permet de transformer une valeur de type `int` en chaîne de caractères.
- La procédure `print` permet d'afficher une chaîne de caractères sur le terminal.

Exercice 12 (Utilisation de fonctions données, ⋆)

1. *On considère la fonction suivante :*

```
def addA (ch) :
    return (ch + "A")
```

Que vaut la variable `st` *après exécution du code suivant ?*

```
s = "Ma lettre finale est "
st = addA (s);
```

2. *Quelle est la valeur de la variable* `a` *après les instructions suivantes ?*

```
ch = "Ceci est "
ch2 = "une chaine"
ch = ch + ch2
a = len (ch)
```

□

Parcourir une chaîne de caractères [Cours]

- Il est possible de récupérer les différents caractères d'une chaîne de caractères sous forme de chaîne de caractères.
- À chaque caractère de la chaîne est associée une position.
- Les positions sont numérotées de 0 à $l-1$ où l est la longueur de la chaîne de caractères.
- Par exemple, si l'on a `s="Hello"`, on peut accéder à la première lettre qui est la chaîne de caractères `"H"` en faisant `s[0]`. Ainsi on a `s[1]` qui vaut `"e"`, `s[2]` qui vaut `"l"`, etc, et `s[5]` n'est pas défini car la chaîne de caractères stockée dans la variable `s` est de longueur 5, par conséquent on ne peut accéder qu'aux caractères aux positions 0, 1, 2, 3 et 4.
- Si l'on cherche à accéder à une position dans une chaîne de caractères supérieure ou égale à la longueur de la chaîne, une erreur sera produite à l'exécution.

Exercice 13 (Parcourir une chaîne de caractères, ⋆)

1. *Que vaut la variable* `cha` *dans l'exemple suivant ?*

```
ch = "Avec consonnes"
cha = ch[0]
cha = cha + ch[2]
```

```
cha = cha + ch[6]
cha = cha + ch[9]
cha = cha + ch[12]
```

□

Exercice 14 (Modification d'une fonction, ⋆⋆)

```
def message (x) :
    return str (x)
```

Et on considère la liste d'instructions suivante :

```
a = 4
b = a * a
a = b - a
s = message (a)
print (s)
```

1. *Quelles sont les valeurs des variables de ce programme à la fin de son exécution ?*
2. *Qu'affichent ces instructions ?*
3. *Est-il possible de modifier la fonction "*`message`*" de telle sorte que le programme affiche à la fin "Le résultat du calcul est n." (où n est la valeur contenue dans le paramètre transmis à* `message`*) ?*
4. *Définissez une fonction "*`square ( n )`*" qui calcule le carré du nombre donné en paramètre et utilisez-la pour afficher une ligne du type $n * n = n^2$ (où n sera remplacé par sa valeur).*

□

La procédure `print` [Cours]

Quelques détails techniques concernant `print` qui sont parfois utiles à savoir :

- En vérité, la procédure `print` peut prendre un nombre quelconque d'arguments, le cas de zéro arguments inclus.
- Pour appeler une procédure sans arguments il faut quand même écrire les parenthèses, donc dans notre cas : `print()`.
- En général, la procédure `print` parcourt tous ses arguments de gauche à droite et les affiche un après l'autre à l'écran. L'affichage des arguments est séparé par le symbole " " (un espace). Après avoir affiché tout ses arguments, la procédure `print` fait un saut à la ligne suivante.
- Par conséquent, `print()` fait simplement un saut au début de la ligne suivante.
- Si on veut éviter le saut de ligne final il faut appeler `print` avec un paramètre supplémentaire précisant ce que l'on veut afficher à la fin. Typiquement le paramètre utilisé pour éviter le retour à la ligne sera `end=""`. Ainsi si l'on fait `print ("Hello World!")`, l'affichage ira à la ligne après avoir affiché `Hello World!` et pour éviter ce retour à la ligne, on pourra faire `print ("Hello World!", end="")`.
- On peut également modifier ce qui est affiché entre les arguments, en écrivant un autre paramètre supplémentaire `sep=s` où s est la chaîne que vous souhaitez afficher entre les arguments.

Exercice 15 (Comprendre les paramètres de print, ⋆⋆)
Quel sera l'affichage produit par les instructions suivantes :

```
print("a", "b")
print("c", "d", end="")
print("e", "f", sep="|")
print("g", "h", end="|")
print("i", "j", end="|\n")
print("k", "l", sep="\n")
```

```
7 print(end="")
```

code/exoPrint.py

□

6 Boucles et chaines de caractères

Pour certains exercices, un "contrat" est proposé : ce sont des tests que nous vous fournissons pour vérifier votre réponse. Si votre réponse ne passe pas ces tests, il y a une erreur ; si votre réponse passe les tests, rien n'est garanti, mais c'est vraisemblablement proche de la réponse. Quand aucun contrat n'est spécifié, à vous de trouver des tests à faire.

Exercice 16 (Pendu à l'envers, ⋆⋆)

1. *Écrire une procédure qui prend en paramètre une chaîne de caractères et affiche une suite de tirets (-) de même longueur.*

   Contrat:
   paramètre = "amphitheatre" → *affichage : - - - - - - - - - - - -*

2. *Écrire une procédure qui prend en paramètre une chaîne de caractères et l'affiche à l'envers.*

   Contrat:
   paramètre = "amphitheatre" → *affichage : ertaehtihpma*

□

Exercice 17 (Affichage sans des espaces, ⋆)
Écrire une procédure qui prend en paramètre une chaîne de caractères et affiche cette même chaîne sans les espaces.

Contrat:
paramètre = "Bonjour monde !" → *affichage : Bonjourmonde !*

□

7 Do it yourself

Exercice 18 (Concaténation, ⋆)
Écrire une procédure qui prend en paramètre une chaîne de caractères `toto` *et affiche une ligne commençant par* `bonjour,` *suivi du contenu de* `toto`.

Contrat:
si `toto` *a la valeur* `"toi"`*, l'appel à la fonction doit afficher*
`bonjour, toi`

□

Exercice 19 (Concaténation et somme, ⋆⋆)
Que valent les variables `x` *et* `s` *après les instructions suivantes ?*

```
1 x = 0
2 s = ""
3 for n in range (0, 10, 1) :
4     x = x + 1
5     s = s + "1"
```

□

Exercice 20 (Condition et division entière, ⋆)
Écrire une condition permettant de tester si le tiers de l'entier `x` *appartient à l'intervalle* $[2;8]$.

Contrat:
x = 0 → *False*
x = 6 → *True*
x = 9 → *True*
x = 25 → *False*

□

Exercice 21 (Condition, ⋆)
Écrire une condition permettant de tester si les entiers `a`, `b` *et* `h` *peuvent correspondre aux longueurs des côtés d'un triangle rectangle, où* `h` *serait la longueur de l'hypothénuse.*

Contrat:
(a,b,h) = (3,4,5) → *True*
(a,b,h) = (1,2,3) → *False*

□

Exercice 22 (Somme des cubes, ⋆)
Écrire une boucle permettant de calculer la somme des cubes des entiers de 1 à 100. □

Exercice 23 (Ordinaux anglais, ⋆⋆⋆)
On s'intéresse aux ordinaux anglais abrégés, *où le nombre (le cardinal) est écrit en chiffres :*

1st, 2nd, 3rd, 4th,..., 9th, 10th, 11th, 12th,..., 19th, 20th, 21st, 22nd, 23rd,...

Pour déterminer le suffixe, on regarde le dernier chiffre du nombre : si c'est 1, on ajoute le suffixe "st" ; si c'est 2, le suffixe est "nd" ; si c'est 3, le suffixe est "rd" ; sinon le suffixe est "th". Il y a cependant une exception : si l'avant-dernier chiffre du nombre est 1, le suffixe est toujours "th".

Écrire une procédure `printOrdinal` *qui prend un entier de type* `int` *en paramètre et affiche l'ordinal anglais abrégé correspondant.*

Contrat:
1 → *1st*
12 → *12th*
23 → *23rd*
32 → *32nd*
44 → *44th*

□

Exercice 24 (Simplification de code, ⋆)

1. *Dire, en justifiant, ce que fait la fonction suivante selon les valeurs entières données aux variables* x *et* y.

```
def f (x, y) :
  if (((x <= y) or (x >= y + 1))) :
    if ((x <= y) and (x < y)) :
      if (x < -x) :
        a = -x
        return a
      else :
        return x
    else :
      if (y * y * y < 0) :
        return -y
  return y
```

2. *Écrire la même fonction de manière plus simple.*

□

Exercice 25 (Mention, ⋆)
Écrire une conditionnelle afin de stocker dans une chaîne de caractères `s` *la mention correspondant à la note (entière) contenue dans la variable* `n` *de type* `int`.

Rappel : entre 10 inclus et 12 exclu, la mention est passable *; entre 12 inclus et 14 exclu ;* bien *entre 14 inclus et 16 exclu, et* très bien *au-delà de 16.* □

Exercice 26 (Lignes et pointillés, ⋆⋆)

1. *Écrivez une procédure qui prend en paramètre un entier supposé positif n et qui affiche une ligne d'astérisques « * » de longueur n puis va à la ligne. Par exemple, si n vaut 7, votre fonction affichera :*
   ****
2. *Modifiez votre procédure pour qu'elle affiche une ligne pointillée alternant astérisques et espaces. Par exemple, si n vaut 7, votre fonction affichera :*
   * * * *
   Qu'affiche votre procédure si n est pair ?

□

Exercice 27 (Nombres parfaits, ⋆⋆⋆)
Un entier $n > 1$ est dit parfait *s'il est égal à la somme de ses diviseurs propres (c'est-à-dire autres que lui-même). Par exemple, 6 est parfait car ses diviseurs propres sont 1, 2 et 3, et on a $6 = 1 + 2 + 3$.*

1. *Écrire une procédure* `divisors` *qui prend en paramètre un entier* `n` *et affiche tous les entiers strictement inférieurs à* `n` *qui divisent* `n`.

   Contrat:
   n = 1 → affichage :
   n = 5 → affichage : 1
   n = 60 → affichage : 1 2 3 4 5 6 10 12 15 20 30

2. *Écrire une procédure* `isPerfect(n)` *qui prend en paramètre un entier* `n` *et qui affiche* `"n est parfait "` *si* `n` *est parfait.*

3. *Écrire une procédure* `enumPerfect(m)` *qui prend en paramètre un entier* `m` *et affiche pour tous les nombres parfaits tels que* `n`≤`m`, `"n est parfait"` *(en remplaçant avec la valeur de* `n`*).*

□

Exercice 28 (Occurrences d'un caractère, ⋆⋆)
Écrire une procédure `inString` *qui prend en paramètre une chaîne de caractères* `s` *et un caractère* `c` *(c'est-à-dire* `c` *est une chaîne de caractères qu'on suppose de longueur 1), et affiche toutes les positions de* `s` *où* `c` *apparaît.*

Contrat:
(s,c) = ("abracadabra","a") → affichage : 0 3 5 7 10

□

Introduction à la Programmation 1 PYTHON

MT11Y060

Séance 3 de cours/TD

Université de Paris

Objectifs:

— Manipuler des boucles avec accumulateurs

— Manipuler deux boucles imbriquées

1 Accumuler des valeurs grâce aux boucles

Utilisation d'accumulateurs dans les boucles [COURS]

— Si une variable est initialisée avant une boucle, sa valeur persiste d'une itération de la boucle à la suivante. On peut donc modifier sa valeur à chaque itération de boucle, et donc *accumuler* une valeur dans cette variable.

— Par exemple, le programme suivant calcule la somme des carrés des entiers de 1 à 100 dans `s` :

```
s = 0
for i in range (1, 101, 1) :
    s = s + i * i
```

Exercice 1 (Accumulation, ⋆)
Modifier la boucle donnée ci-dessus pour qu'elle calcule la somme des cubes des entiers de 1 à 42. □

Exercice 2 (Comprendre et modifier une boucle, ⋆)

```
st = "";
for i in range (1, 51, 1) :
    st = st + "ab"
```

1. *Que contient la variable* `st` *après la suite d'instructions donnée ci-dessus ?*
2. *Modifier les instructions ci-dessus pour qu'à la fin de leur exécution la variable* `st` *contienne la chaîne de caractère "aaaaaa ... aaaa" avec la lettre "a" répétée 110 fois.*
3. *Modifier de nouveau la suite d'instructions pour imprimer à l'écran à chaque tour de boucle le contenu de la variable* `st`*. Qu'affichera l'exécution de votre code ?*

□

Exercice 3 (Accumulation booléenne, ⋆⋆)
L'expression `input ()` *permet de lire une chaîne de caractères tapée au clavier et de la renvoyer. Si on veut lire un entier tapé au clavier, il suffit de convertir cette chaîne en entier en faisant* `int (input ())`.

Écrire un programme qui lit 5 entiers au clavier, puis affiche « Gagné » si l'entier 42 se trouvait parmi ceux-là, et sinon « Perdu ». Attention, il faut lire les 5 entiers et seulement ensuite afficher le résultat. □

2 Boucles imbriquées

Boucles imbriquées [Cours]

Le corps d'une boucle peut lui-même contenir une boucle. On parle alors de *boucles imbriquées*. Ainsi, le programme :

```
for i in range (0, 3, 1) :
  for j in range (1, 5, 1) :
    print (" i = " + str(i) + ", j = " + str(j))
```

affiche :

```
 i = 0, j = 1
 i = 0, j = 2
 i = 0, j = 3
 i = 0, j = 4
 i = 1, j = 1
 i = 1, j = 2
 i = 1, j = 3
 i = 1, j = 4
 i = 2, j = 1
 i = 2, j = 2
 i = 2, j = 3
 i = 2, j = 4
```

- À chaque tour de la boucle externe, la boucle interne parcourt toutes les valeurs de 1 à 4.
- La boucle externe est lente, la boucle interne est rapide.
- Chaque boucle a son propre compteur.

Exercice 4 (Cent, ⋆)
Écrire un programme qui affiche les nombres de 0 *à* 99*, à raison de* 10 *nombres par ligne :*

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13...
...
90 91 92 93 94 95 96 97 98 99
```

□

Exercice 5 (Triangle, ⋆⋆)
Écrire une procédure qui prend en paramètre un entier n *et affiche, pour* i *allant de* 1 *à* n*,* i *étoiles sur la* i*-ième ligne. Par exemple, pour* $n = 5$*, afficher :*

```
*
* *
* * *
* * * *
* * * * *
```

□

3 Do it yourself

Exercice 6 (Puissance, $\star$)
Écrire une fonction "power (x, n)" qui renvoie la valeur x^n. □

Exercice 7 (Factorielle, $\star$)
Écrire une fonction "`fact (n)`*" qui renvoie la factorielle de n.* □

Exercice 8 (Ça use, $\star\star$)

1. *Écrire un programme qui affiche les paroles de la chanson que vos neveux chantaient durant les vacances d'été :*
   ```
   1 kilomètre à pied, ça use les souliers.
   2 kilomètres à pied, ça use les souliers.
   ...
   100 kilomètres à pied, ça use les souliers.
   ```
   Attention, le premier vers est différent (le mot kilomètre *est au singulier).*
2. *Modifier votre code pour qu'il affiche les vers dans l'ordre inverse.*

□

Exercice 9 (Nombres premiers, $\star\star$)

1. *Écrire une fonction "*`isPrime (n)`*" qui renvoie 1 si n est un nombre premier, 0 sinon. (On rappelle que 1 n'est* pas *un nombre premier ; le plus petit nombre premier est donc 2.)*
2. *Écrire une fonction "*`sumPrime (n)`*" qui renvoie la somme des nombres premiers compris (au sens large) entre 1 et n.*

□

Exercice 10 (Table de multiplication, $\star\star$)

1. *Écrire un programme qui affiche les tables de multiplication pour les entiers de 1 à 10.*
   ```
   1 2 3 4 ... 10
   2 4 6 8 ... 20
   ...
   10 20 30 ... 100
   ```
2. *Écrire une fonction qui prend un entier n en paramètre et renvoie 1 si n apparaît dans une des tables de multiplication de 1 à 10, et 0 sinon.*

□

Exercice 11 (Carrés, $\star\star$)

1. *Écrire une procédure qui prend en paramètre un entier positif n et affiche un carré d'étoiles plein de côté n. Par exemple, si n vaut 4, votre fonction affichera*
   ```
   ****
   ****
   ****
   ****
   ```
2. *Modifier votre procédure pour qu'elle affiche un carré creux, par exemple pour n valant 4,*
   ```
   ****
   *  *
   *  *
   ****
   ```

□

Exercice 12 (Séries, $\star$ – $\star\star\star$)
Pour chacune des séries suivantes, trouver le plus petit programme (en nombre d'appels de fonctions) qui affiche les séries de 20 nombres suivants à l'écran :

- *0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 ($\star$)*
- *1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 ($\star$)*
- *0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 ($\star\star\star$)*
- *0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 ($\star\star$)*
- *0 2 4 6 8 9 11 13 15 17 18 20 22 24 26 27 29 31 33 35 ($\star\star\star$)*

□

Exercice 13 (Sous-chaîne, $\star\star$)
Écrire une procédure `sousChaine` qui prend en paramètre deux chaînes de caractères `s` et `ss` non vides, et affiche toutes les positions de `s` à partir desquelles `ss` apparaît comme sous-chaîne.

Contrat:
(s,ss) = ("abracadabra","abra") $\rightarrow$ affichage : 0 7

□

Introduction à la Programmation 1 PYTHON

MT11Y060

Séance 4 de cours/TD

Université de Paris

Objectifs:

- Déclarer et initialiser une liste de type `int`.
- Écrire des fonctions qui attendent des listes en entrée et produisent des listes en sortie.

1 Les listes d'entiers

Listes d'entiers [COURS]

- Une liste est une suite ordonnée de valeurs.
- Une liste d'entiers contenant 2 comme première valeur, 4 comme deuxième valeur et 12 comme troisième valeur sera notée `[2, 4, 12]`.
- La liste vide est notée `[]`.
- Comme toute valeur, une liste peut être stockée dans une variable. De même, une fonction peut prendre une liste en paramètre et renvoyer une liste.
- Le nombre d'éléments d'une liste est appelé sa *longueur*.
- Si n est la longueur d'une liste alors les valeurs de la liste sont numérotées de 0 à $n-1$. Ces numéros sont appelés *indices*.
- Pour lire la première valeur de la liste `l`, on écrit « `l[0]` ».
  Plus généralement, pour lire la valeur d'indice `i`, on utilise l'expression « `l[i]` ».
- Pour modifier la i-ème valeur d'une liste `l`, on utilise l'instruction « `l[i] = e` » où e est une expression qui s'évalue en une valeur.
- **Attention** à ne jamais lire ou écrire hors des limites de la liste car cela produit une erreur. Ainsi, les indices négatifs ou plus grand que la longueur de la liste, ou égaux à la longueur de la liste, sont à éviter.
- La fonction `len` permet d'obtenir la longueur d'une liste. Par exemple `len ([1, 2 , 7, 5])` vaut 4.
- On peut afficher une liste en utilisant la procédure d'affichage `print`, cette fonction appelée avec une liste comme paramètre affiche sur une ligne le contenu de la liste entre crochets. Attention toutefois, si l'on veut afficher la valeur d'indice 2 de la liste `l`, il faut faire `print (l[2])`.

Exercice 1 (Comprendre la taille et les indices, $\star$)

1. *Si une liste* `a` *vaut* $[1,3,5,7,9,11,13,15,17]$. *Quelle est sa taille ? À quel indice trouve-t-on la valeur* 1 *? À quel indice trouve-t-on la valeur* 17 *? À quel indice trouve-t-on la valeur* 9 *?*
2. *Si une liste* `b` *vaut* $[2,2,3,3,4,5,7]$. *Quelle est sa taille ? Que vaut l'expression* `b[0]` *? Que vaut l'expression* `b[4]` *? Pourquoi ne peut-on pas évaluer* `b[7]` *?*

□

Exercice 2 (Listes et expressions, ⋆⋆)
Si la liste `l` *vaut* `[1,2,4]`*, quelle sont les valeurs des expressions suivantes :*

1. `l[l[0]]`
2. `l[l[2]-l[1]]`

□

Exercice 3 (Lecture, ⋆)
La déclaration suivante introduit une liste `l` *dont les valeurs sont de type* `int` *et qui vaut* $[2, 3, 4, 5, 6, 7]$.

```
l = [2, 3, 4, 5, 6, 7]
```

1. *Donner une instruction modifiant la deuxième valeur de la liste* `l` *pour y mettre la valeur* 10.
2. *En utilisant la procédure* `print` *pour afficher la valeur d'un entier, donner les instructions permettant d'afficher la première valeur de la liste* `l` *(celle à l'indice* 0*) et la dernière valeur de* `l`.
3. *Quel est le contenu de la liste* `l` *après exécution des instructions suivantes :*

```
l = [2, 3, 4, 5, 6, 7]
for i in range (0, len(l), 1) :
    l[i] = i
```

□

Exercice 4 (Swap, ⋆)
On considère la déclaration suivante :

```
a = [12, -3, 22, 55, 9]
```

1. *Donner une suite d'instructions permettant d'inverser la première et la deuxième valeur de la liste* `a`.
2. *Donner une suite d'instructions permettant d'inverser la première et de la dernière valeur de la liste* `a`.

□

2 Opérations sur les listes

Concaténation de liste [Cours]

— Il est possible de regrouper deux listes en une seule grâce à l'opérateur +. Par exemple, `[4, 3] + [1, 2]` donne la liste `[4, 3, 1, 2]`. Bien entendu cet opérateur peut être utilisé avec des variables contenant des listes ou plus généralement entre toutes expressions s'évaluant en des listes.
— On peut aussi créer une liste en répétant une autre liste grâce à l'opérateur binaire ∗. Ainsi l'opération `[1, 2] * 3` donne la liste `[1, 2, 1, 2, 1, 2]` où l'on a répété la liste `[1, 2]` trois fois. Cette instruction permet d'initialiser par exemple une liste avant de la remplir.
— Par exemple, le morceau de code suivant crée la liste `a = [-2, -1, 0, 1]`.

```
a = [0] * 4
a[0] = -2
a[1] = -1
a[2] = 0
a[3] = 1
```

Exercice 5 (Création de listes, ⋆)
Donner les instructions pour créer une liste `a` *valant* $[1, 2, 3, 4, 5, \ldots, 1000]$.
(Il est bien sûr recommandé d'utiliser une boucle `for` *pour remplir un telle liste.)* □

Exercice 6 (Indices, ⋆)
Créer une liste `a` *de longueur* 10 *et dont chaque élément se trouvant à l'indice* i *a pour valeur* $2 * i$. □

Exercice 7 (Manipulation de listes, ⋆)
Donner les instructions réalisant les opérations suivantes :

1. *Créer une liste* `l` *valant* $[2, 4, 6, 8, 10, 12, 14, 16]$.
2. *Écrire une fonction qui prend une liste* `l` *en paramètre, et qui renvoie une nouvelle liste de la même taille que* `l` *qui contient à chaque indice le double de la valeur contenue dans* `l` *à l'indice correspondant.*

□

Exercice 8 (Liste d'éléments non nuls, ⋆⋆)

1. *Que fait la fonction «* `notZero (l)` *» dont le code vous est donné ci-dessous :*

```
def notZero (l) :
  n = len (l)
  s = 0
  for i in range (0, n, 1) :
    if (l[i] != 0) :
      s = s + 1
  return s
```

2. *Donner une séquence d'instructions réalisant les actions suivantes :*
   (a) *Création d'une liste* `l` *valant :* $[4, 3, 6, 2, 0, 0, 2, 0, 4]$.
   (b) *Création d'une liste* `l2` *contenant les éléments de la liste* `l` *qui sont différents de zéro (dans le même ordre). Pour cela vous utiliserez la fonction* `notZero`.

□

3 Do it yourself

Exercice 9 (Somme, ⋆)

Écrire une fonction « `sumIntList (l)` *» qui calcule la somme des éléments de la liste d'entiers* `l` *donnée en paramètre.* □

Exercice 10 (Moyenne, ⋆)

Écrire une suite d'instructions qui crée une liste d'entiers `l` *valant* `[3, 5, 2, 3, 6, 3, 4]`*, qui calcule la moyenne entière des éléments de* `l` *et qui affiche cette moyenne.*

□

Exercice 11 (Grognements, ⋆)

Écrire une procédure qui prend un paramètre entier `n` *et affiche les* `n` *premiers grognements* `brhh`, `brrhh`, `brrrhh`, *etc.*

□

Exercice 12 (Dernière occurrence, ⋆⋆)

Écrire une fonction « `lastOcc (l, x)` *» qui prend en paramètre une liste d'entiers et une valeur entière. Si la liste contient cette valeur, la fonction renvoie l'indice de la dernière occurrence de la valeur. Si la liste ne contient pas cette valeur, la fonction renvoie* -1.

□

Exercice 13 (Première occurence, ⋆⋆)

Écrire une fonction « `firstOcc (l, x)` *» qui prend en paramètre une liste et une valeur. Si la liste contient cette valeur, la fonction renvoie l'indice de la* première *occurrence de la valeur. Si la liste ne contient pas cette valeur, renvoie* -1.

Indice : on pourra tester si le résultat à renvoyer est différent de -1 pour comprendre si l'on a déjà vu la valeur dans la liste.

□

Exercice 14 (Pyramide, ⋆⋆⋆)

Écrire une procédure qui prend en entrée un entier n *et affiche une pyramide d'étoiles de hauteur* n. *Si* n *est négatif, la pyramide devra être inversée. Par exemple, sur entrée* $n = 4$, *afficher :*

```
   *
  * *
 * * *
* * * *
```

Sur entrée $n = -3$, *afficher :*

```
* * *
 * *
  *
```

□

Exercice 15 (Suite dans une liste, ⋆⋆)

Écrire une fonction « `progression (a, b, n)` *» qui prend en paramètre trois entiers* `a`, `b` *et* `n`, *et qui renvoie une liste de taille* `n` *contenant les entiers* `a, a + b, a + 2b, a + 3b, ..., a + (n-1)b`.

□

Exercice 16 (Entrelace, ⋆⋆)

Écrire une fonction « `interlace(l1, l2)` *» qui prend en entrée deux listes* `l1` *et* `l2` *de même longueur et renvoie une liste de longueur double qui contient les valeurs des listes de façon entrelacée, c'est-à-dire* `[l1[0], l2[0], l1[1], l2[1], ..., l1[n], l2[n]]` *(*`n` *est ici la longueur des deux listes). Par exemple, appliquée à* `[0, 1, 6]` *et* `[2, 4, 7]`, *elle va renvoyer la liste* `[0, 2, 1, 4, 6, 7]`.

□

Exercice 17 (Plagiat, ⋆⋆)

1. *Écrire une fonction* `plagiarism` *prenant en paramètre deux listes* `l1` *et* `l2` *et qui renvoie l'indice d'une valeur de* `l2` *qui apparaît dans* `l1`. *Si une telle valeur n'existe pas, la fonction renvoie* -1.
2. *Que se passe-t-il si on appelle la fonction* `plagiarism` *sur la même liste* `plagiarism (l, l)` *?*
3. *Écrire une fonction* `autoPlagiarism` *prenant en paramètre une liste* `l` *et qui renvoie l'indice d'une valeur de* `l` *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1.

□

Exercice 18 (Récurrence, ⋆⋆⋆)

Soient U *et* V *deux listes d'entiers, de même taille arbitraire* k. *On considère l'équation de récurrence suivante :*

$$u_n = U[n] \text{ pour } 0 \leq n < k \quad \text{et} \quad u_{n+1} = \sum_{i=0}^{k-1} V[i]u_{n-i} \text{ pour } n \geq k$$

Écrire une fonction qui prend en paramètre deux listes d'entiers U et V (qu'on supposera de même taille) et un entier n, et qui renvoie la valeur de u_n définie ci-dessus. Attention, on ne connaît pas à l'avance la taille des listes. □

Exercice 19 (Mastermind, ⋆⋆⋆)

Dans cet exercice, on se propose d'écrire un programme permettant de jouer au Mastermind©.

On rappelle le principe de ce jeu. Un joueur (ici, la machine) choisit une **combinaison secrète** *de 5 couleurs parmi 8 couleurs disponibles (***on représentera durant tout l'exercice les couleurs par les entiers 0, 1, 2, 3, 4, 5, 6, 7***). Une couleur peut apparaître plusieurs fois dans une combinaison. L'adversaire de ce joueur a 10 tentatives pour deviner cette combinaison sachant qu'après chaque proposition, il bénéficie d'une indication : le premier joueur examine sa proposition et lui dit combien de pions colorés sont bien placés et combien sont présents dans la combinaison secrète, mais mal placés.*

Ainsi dans l'exemple suivant :

combinaison secrète	$[4, 2, 6, 0, 5]$
combinaison proposée	$[1, 6, 4, 0, 5]$

le joueur qui devine se verra répondre "2 pion(s) bien placé(s) et 2 pions mal placé(s)", correspondant aux couleurs 0 et 5 pour les bien placées, et aux couleurs 6 et 4 pour les mal placées.

Voici un deuxième exemple pour bien fixer les idées :

combinaison secrète	$[1, 1, 6, 0, 3]$
combinaison proposée	$[1, 6, 4, 1, 1]$

le joueur qui devine se verra répondre "1 pion(s) bien placé(s) et 2 pion(s) mal placé(s)", correspondant à la couleur 1 en position 0 bien placée et les couleurs 1 (position 3 ou 4) et 6 (position 1).

- *Écrire une fonction « `wp (prop, sol)` » qui renvoie le nombre de pions bien placés, si `sol` est une liste représentant la solution et `prop` la proposition.*
- *Écrire une fonction « `count (l)` » qui renvoie une liste de taille 8 dont la i-ème case contient le nombre d'occurrences de la valeur i dans la liste d'entiers `l`..*
- *On veut désormais écrire une fonction permettant de compter le nombre de pions mal placés en se basant sur l'observation suivante. Pour une couleur donnée, si elle apparaît x fois dans la proposition et y fois dans la solution secrète, alors le nombre de pions mal placés de cette couleur plus le nombre de pions bien placés de cette couleur est égal au minimum de x et de y.*
  En déduire une fonction « `bp(prop, sol)` » qui renvoie le nombre de couleurs mal placées. On pourra écrire une fonction « `minInt (x, y)` » qui renvoie le minimum des deux entiers `x` et `y` donnés en paramètre.
- *On suppose que l'on a à notre disposition une fonction « `randSol ()` » qui renvoie une liste aléatoire de longueur 5 constitué de valeurs de 0 à 7.*
  Écrire maintenant un programme qui simule le jeu, c'est-à-dire qu'il laisse à l'utilisateur dix tentatives pour trouver la solution en lui donnant après chaque tentative le nombre de bien placés et de mal placés. Si le joueur a trouvé la bonne solution, la boucle doit s'arrêter en affichant un message de victoire, et si le joueur échoue après dix tentatives, le programme doit afficher un message d'échec en lui donnant la combinaison secrète. A chaque tour, l'utilisateur entrera sa combinaison en tapant les 5 chiffres successivement. Pour cela, on pourra utiliser l'instruction `int (input ())` qui permet de récupérer un entier saisi au clavier par l'utilisateur.

□

Exercice 20 (Réordonnancement, ⋆⋆)

Écrire une procédure qui demande un entier n à l'utilisateur, puis lui demande n fois d'entrer un entier x ainsi qu'une position $j \in \{1, \ldots, n\}$ (pour l'affichage), et qui affiche ensuite les n entiers selon l'ordre spécifié. Ainsi, si l'utilisateur entre

```
4
10
3
12
2
15
4
16
1
```

(la première ligne correspondant au nombre d'entiers à entrer, puis les lignes paires sont les entiers eux-mêmes et les lignes impaires leur position), il souhaite afficher l'entier 10 en position 3, l'entier 12 en position 2, 15 en position 4 et 16 en première position. Le programme devra alors afficher :

16 12 10 15

Pour cela, on pourra utiliser l'instruction `int (input ())` *qui permet de récupérer un entier saisi au clavier par l'utilisateur (après que celui-ci ait tapé "Entrée")* □

Introduction à la Programmation 1 PYTHON

MT11Y060

Séance 5 de cours/TD

Université de Paris

Objectifs:

- Comprendre les listes de chaînes de caractères.
- Comprendre les listes de listes
- Comprendre le statut particulier des listes en mémoire.

1 Listes d'autres types

Des listes de chaînes de caractères [COURS]

- Une liste peut contenir aussi des chaînes de caractères.
- Par exemple, pour créer une liste de chaînes de caractères `theBeatles` qui vaut initialement `["Paul", "John", "Ringo","Georges"]`, on peut procéder ainsi :

```
theBeatles = ["Paul", "John", "Ringo", "Georges"]
```

ou bien ainsi :

```
theBeatles = [""]*4
theBeatles[0] = "Paul"
theBeatles[1] = "John"
theBeatles[2] = "Ringo"
theBeatles[3] = "Georges"
```

Exercice 1 (Fonctions et liste de chaînes de caractères, ⋆)

1. *Que fait la fonction de signature « `funcAB (a)` » fournie ci-dessous en supposant qu'elle prend en paramètre un entier strictement positif ?*

```
def funcAB (a) :
  l = [""] * a
  s = "ab"
  for i in range (0, a, 1) :
    l[i] = s
    s = s + "ab"

  return (l)
```

2. *Utilisez la fonction précédente dans une suite d'instructions pour afficher 5 lignes de la forme suivante :*

```
ab
abab
ababab
abababab
ababababab
```

(On pourra utiliser la procédure « `print` ».)

□

Exercice 2 (Listes de prénoms, ⋆⋆)
Écrire une fonction prenant en paramètre une liste de prénoms `l` *et qui renvoie l'indice d'un prénom de* `l` *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1. □

Des listes de listes [Cours]

— Les listes sont des valeurs comme les autres. Une liste peut donc aussi contenir une liste dans chacune de ses cases. On parle alors de liste de listes.
— **Remarque :** En fait les éléments d'une liste ne sont pas nécessairement de même type, mais dans ce cours nous nous imposerons de manipuler et de créer des listes dont tous les éléments ont le même type.
— Par exemple, une liste de liste d'entiers peut valoir « `[[1], [11, 22], [111, 222, 333]]` ». La première valeur de cette liste est la liste « `[1]` » , puis à la deuxième valeur de la liste est « `[11, 22]` » et la dernière valeur de la liste est « `[111, 222, 333]` ».
— Pour créer et initialiser une liste de listes, il y a différentes façons de procéder :

1. On peut créer et initialiser toutes les listes au moment de l'affectation de la liste "contenant" :

```
l = [ [1, 2], [11, 22], [111, 222]]
```

2. On peut créer toutes les listes avec des valeurs initiales égales. **Attention :** Pour faire cela, la liste principale contiendra au début des entiers que l'on remplacera par des listes.

```
l = [0] * 3
l[0] = [0] * 5
l[1] = [0] * 5
l[2] = [0] * 5
```

L'exemple précédent crée une liste de 3 valeurs, et chaque valeur est une liste de 5 valeurs entières. Dans cet exemple toutes les listes contenues dans la première liste ont la même taille 5, mais on aurait pu leur donner des tailles différentes comme dans l'exemple suivant :

```
li = [0] * 3
li[0] = [0] * 1
li[1] = [0] * 2
li[2] = [0] * 3
```

3. Pour extraire un élément d'une liste de listes il suffit d'enchaîner les indices, d'abord l'indice dans la liste de listes, puis l'indice dans la liste d'entiers sélectionnée. Par exemple :

```
l = [ [0], [1, 2], [3, 4] ]
print(l[2][0])

```

affiche la valeur 3.

4. Pour modifier les valeurs des listes "contenues", on peut utiliser des instructions de modification du contenu de listes en précisant leurs indices :

```
l = [ [0], [1, 2], [3, 4] ]
l[2][0] = 15
```

Cette instruction met la valeur 15 au premier indice de la troisième liste (la liste qui se trouve à la fin de la liste de listes l).
La déclaration suivante crée la liste « `liPascal` » valant « `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1]]` » :

```
liPascal = [0] * 4
liPascal[0] = [0]
liPascal[0][0] = 1
liPascal[1] = [0] * 2
liPascal[1][0] = 1
liPascal[1][1] = 1
liPascal[2] = [0] * 3
liPascal[2][0] = 1
liPascal[2][1] = 2
liPascal[2][2] = 1
liPascal[3] = [0] * 4
liPascal[3][0] = 1
liPascal[3][1] = 3
liPascal[3][2] = 3
liPascal[3][3] = 1
```

— On parlera de listes à deux dimensions (ou bi-dimensionnelle) pour des listes de listes, les listes à trois dimensions seront des listes de listes de liste, etc.

Exercice 3 (Table de multiplication, $\star$)

1. *Créer une liste à deux dimensions tel que* `l[i][j]` *vaut* `(i + 1) * (j + 1)` *pour* i *et* j *allant de* 0 *à* 9.
2. *Où se trouve le résultat de la multiplication de* 3 *par* 4 *dans cette liste ? Même question pour le résultat de la multiplication de* 7 *par* 6.

□

Exercice 4 (Carré magique, $\star\star\star$)
Un carré magique est une grille carrée dans laquelle des nombres sont placés de telle sorte que la somme des nombres de chaque colonne, chaque ligne et chacune des deux diagonales soit la même. De plus, le carré doit contenir une fois chaque nombre, de 1 au nombre de cases de la grille. La grille peut être représentée comme une liste bi-dimensionnelle d'entiers. Notre objectif est d'écrire une fonction qui vérifie si une grille de nombres reçue comme paramètre est un carré magique.

1. *Écrire une fonction* `carre` *qui prend une liste de listes d'entiers* `m` *en paramètre et qui vérifie que* `m` *représente bien une grille carrée.*
2. *Écrire une fonction* `aplatir` *qui prend en paramètre une liste de listes d'entiers* `m` *qu'on peut supposer être une grille carrée, et qui envoie une liste d'entiers qui contient tous les entiers présents dans* `m`. *Par exemple, appliquée à* `[[1, 2, 3], [4, 5, 6], [6, 8, 9]]`, *la fonction doit envoyer le résultat* `[1, 2, 3, 4, 5, 6, 6, 8, 9]`.
3. *Écrire une fonction* `domaine` *qui prend une liste d'entiers* `l` *en paramètre et qui envoie le booléen* `True` *si tous les éléments de* `l` *sont des valeurs entre* 0 *(non-inclus) et la longueur de la liste (incluse), et*

`False` *sinon.*

4. *Écrire une fonction* `differents` *qui prend une liste d'entiers* `l` *en paramètre et qui envoie le booléen* `True` *si tous les éléments de* l *sont des valeurs différentes et* `False` *sinon.*
5. *Écrire une fonction* `lignes` *qui prend une liste de listes d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque ligne est égale à* `a` *et* `False` *sinon.*
6. *Écrire une fonction* `colonnes` *qui prend une liste de listes d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque colonne est égale à* `a` *et* `False` *sinon.*
7. *Écrire une fonction* `diagonales` *qui prend une liste de listes d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque diagonale est égale à* `a` *et* `False` *sinon.*
8. *Enfin, écrire une fonction* `magique` *qui prend une liste de listes d'entiers* `m` *en paramètre et qui envoie* `True` *si* `m` *représente un carré magique, et* `False` *sinon. Utilisez les fonctions demandées aux questions précédentes.*

□

Exercice 5 (Léger Gomoku , ⋆⋆⋆)
Le Gomoku est un jeu de plateau à deux joueurs, dans lequel pour gagner, chaque joueur doit réussir à aligner 5 pions sur des cases consécutives d'un plateau, horizontalement, verticalement ou en diagonale. On va faire une version légère où **on ne prendra pas en compte les diagonales**. *Le plateau est une grille carrée de taille quelconque supérieure ou égale à 5, et il peut être représenté comme une liste de listes d'entiers. L'entier vaut 0 si la case est vide, 1 si elle contient un pion du joueur 1, et 2 pour un pion du joueur 2. On souhaite écrire une fonction* `winGomoku` *qui reçoit comme paramètre le contenu d'une partie de Gomoku et affiche* `"Personne ne gagne"`, *ou* `"Joueur 1 gagne"` *ou* `"Joueur 2 gagne"` *ou* `"Mauvaise grille"`, *selon les cas.*

1. *Écrire une fonction* `isGomokuSquare` *qui teste si la liste de listes d'entiers donnée en paramètre correspond à une grille carrée de dimension supérieure ou égale à 5, c'est-à-dire chaque liste de la liste principale doit avoir la même longueur que la liste principale et cette longueur doit être supérieure ou égale à 5.*
2. *Écrire une fonction* `fiveInARow` *qui prend en paramètre une liste de listes d'entiers correspondant à une grille carrée et un numéro de joueur (qui vaudra 1 ou 2) et qui teste si une liste de la grille carrée contient cinq fois le numéro du joueur à la suite.*
3. *Écrire une fonction* `fiveInAColumn` *qui prend en paramètre une liste de listes d'entiers correspondant à une grille carrée et un numéro de joueur (qui vaudra 1 ou 2) et qui teste si une colonne de la grille carrée contient cinq fois le numéro du joueur à la suite.*
4. *En vous servant des fonctions précédentes, donner le code de la fonction* `winGomoku` *(si les deux joueurs vérifient la condition pour gagner, on suppose que c'est le joueur 1 qui gagne).*

□

2 Le statut particulier des listes en mémoire

Mémoire et liste [Cours]

— Dans la mémoire, on stocke également les listes et leur contenu.
— Si `a` est une variable référant une liste créée, alors dans la mémoire, la variable `a` sera associée à la valeur $\$i$ où i est un entier positif, appelée *son adresse dans le tas*. Le tas est une mémoire auxiliaire *qui est préservée par les appels de fonction et de procédure.*

— Dans notre modèle d'exécution des programmes, le contenu d'une liste est représenté *à côté de la mémoire des variables* dans une autre mémoire. Cette seconde mémoire, le tas, associe des listes à des valeurs de la forme $\$i$.
— Par exemple :

$l1$	$l2$
$\$1$	$\$2$

$\$1 = [2, 3, 4, 5] \quad \$2 = [-9, -3, 0]$

indique une mémoire où la variable `l1` réfère la liste `[2, 3, 4, 5]` et la variable `l2` la liste `[-9, -3, 0]`
— Si une expression accède aux valeurs de la liste (par exemple dans « `a[2]` ») alors il faut aller regarder dans la mémoire quelle est la liste référée par la variable `a`, par exemple $\$3$ et ensuite prendre la valeur qui se trouve dans la troisième case de la liste $\$3$. Le même procédé s'applique pour les modifications du contenu de la liste.
— Si une liste $\$i$ n'est référencée par aucune variable dans la mémoire, alors on peut la supprimer.
— Comme le contenu du tas est préservé par les appels de fonctions et de procédures, on peut écrire des fonctions et des procédures qui modifient les listes passées en paramètres. C'est très pratique de ne pas avoir à recopier le contenu des listes à chaque appel de procédure car la taille des listes peut être importante.

Exercice 6 (Échange du contenu de deux indices, $\star\star$)
Soit la procédure suivante :

```
def swap (a, i, j) :
  tmp = a [i]
  a[i] = a[j]
  a[j] = tmp
```

Donner l'évolution de la mémoire et du tas au cours de l'évaluation des instructions suivantes :

```
li = [ 3, 2, 1, 0 ]
swap (li, 1, 2)
swap (li, 0, 3)
```

□

Exercice 7 (Listes et partage, $\star\star\star$)
On a vu le code suivant pour créer une liste de listes :

```
    li = [0] * 3
    li[0] = [0] * 5
    li[1] = [0] * 5
    li[2] = [0] * 5
```

Il se trouve que la liste de listes créée par le code au-dessus n'est pas la même que la liste de listes créée par le code suivant :

```
    li = [0] * 3
    co = [0] * 5
    li[0] = co
    li[1] = co
    li[2] = co
```

Expliquez la différence. Proposez une expérience qui démontre la différence. □

3 DIY

Exercice 8 (Comptage, ⋆⋆)
Écrire une fonction qui, étant donnée une liste `l` *de nombres entiers et un nombre entier* `n`*, renvoie :*

- *la liste vide si la liste donnée contient (au moins) un nombre* `x` *tel que* $x < 0$ *ou* $x > n$*,*
- *une liste* `li` *de* $n + 1$ *entiers tel que* `li[i]` *soit égal au nombre d'éléments de* `l` *égaux à* `i` *si la liste ne contient que des nombres compris, au sens large, entre* `0` *et* `n`*.*

> ***Contrat:***
> *Si la liste vaut* `[0, 1, 2, 2, 0, 0]` *alors*
> *si le deuxième argument vaut 1, la fonction renverra* `[]`
> *si le deuxième argument vaut 2, la fonction renverra* `[ 3, 1, 2]`

□

Exercice 9 (Suites d'entiers, ⋆⋆)
Toute suite finie d'entiers peut être décomposée de manière unique en une suite de séquences strictement croissantes maximales. En représentant une suite dans une liste `l`*, la liste*

$$[1\ 2\ 5\ 7\ \underline{2}\ 6\ \underline{0}\ 5\ \underline{2}\ 4\ 6\ 7\ 8\ 9\ \underline{3}\ 4\ 6\ \underline{1}\ 2\ 7\ 8\ 9\ \underline{4}\ \underline{2}\ 3\ \underline{1}\ 5\ 9\ \underline{7}\ \underline{1}\ 6\ \underline{6}\ \underline{3}]$$

se décompose ainsi en la liste de 13 listes d'entiers suivante :

$$[\ [1\ 2\ 5\ 7]\ [2\ 6]\ [0\ 5]\ [2\ 4\ 6\ 7\ 8\ 9]\ [3\ 4\ 6]\ [1\ 2\ 7\ 8\ 9]\ [4]\ [2\ 3]\ [1\ 5\ 9]\ [7]\ [1\ 6]\ [6]\ [3]\ ].$$

Les premiers éléments de ces séquences sont, en plus du premier élément de la liste, les éléments (soulignés sur l'exemple) qui sont inférieurs ou égaux à leur prédécesseur dans la liste (`l[i]`$\leq$`l[i-1]`*).*

1. *Écrire une fonction* `numRuptures` *qui à partir d'une liste* `l` *d'entiers donnée en paramètre renvoie le nombre d'indices* `i` *tels que* `l[i]`$\leq$`l[i-1]`*. Pour la liste donnée en exemple, la fonction renvoie 12 (le nombre de positions soulignées).*
2. *Écrire une fonction* `rupture` *qui, étant donnée une liste* `l` *d'entiers, renvoie une liste contenant 0 en premier élément et les indices des éléments de* `l` *inférieurs ou égaux à leur prédécesseur en ordre croissant. Pour la liste donnée en exemple, la fonction renvoie la liste :*

$$[0\ 4\ 6\ 8\ 14\ 17\ 22\ 23\ 25\ 28\ 29\ 31\ 32].$$

3. *Écrire une fonction* `factorisation` *qui, étant donné une liste* `l` *d'entiers, renvoie une liste de listes d'entiers, dont la* `i`*-ème liste contient la* `i`*-ème plus longue séquence croissante de nombres adjacents dans la liste* `l` *(résultat tel que celui donné dans l'exemple). Indication : on pourra utiliser la fonction* `rupture` *pour déterminer le nombre de listes et la taille de chaque liste de cette liste de liste.*

□

Exercice 10 (Matrices, ⋆⋆)
Étant donnée une matrice (liste à deux dimensions) `A` *avec* `n` *lignes et* `m` *colonnes, un couple d'indices (*`i`,`j`*) représente un* min-max *de cette matrice si la valeur* `A[i][j]` *est un minimum de la ligne* `i` *et un maximum de la colonne* `j`*, c'est-à-dire*

$$\texttt{A[i][j]} = min\{\texttt{A[i][0]}, \ldots, \texttt{A[i][m]}\} \quad \texttt{A[i][j]} = max\{\texttt{A[0][j]}, \ldots, \texttt{A[n][j]}\}.$$

Ecrivez une procédure prenant en paramètre une liste de liste d'entiers encodant une matrice et qui affiche l'ensemble de tels couples (`i`,`j`*). La méthode proposée consiste à effectuer le travail suivant pour chaque ligne* `i` *: (1) trouver les minima de la ligne* `i` *et en mémoriser les numéros de colonne et (2) pour chacun de ces rangs* `j`*, déterminer si* `a[i][j]` *est un maximum pour sa colonne.* □

Exercice 11 (Championnat, ⋆⋆⋆)
On considère une liste à 3 dimensions stockant les scores d'un championnat de handball. Pour `n` *équipes, la liste aura* `n` *lignes et* `n` *colonnes et à chacun de ses indices on trouvera une liste de longueur 2 contenant le score d'un match (on ne tient pas compte de ce qui est stocké dans la diagonale). Ainsi, pour la liste*

championnat `ch`*, on trouvera dans* `ch[i][j]` *le score du match de l'équipe* `i+1` *contre l'équipe* `j+1` *et dans* `ch[j][i]` *le score du match de l'équipe* `j+1` *contre l'équipe* `i+1`*. De même, pour un score stocké, le premier entier de* `ch[i][j]` *sera le nombre de but(s) marqué(s) par l'équipe* `i+1` *dans le match l'opposant à l'équipe* `j+1`*. Par conséquent, dans* `ch[i][j][0]` *on trouve le nombre de bus obtenu par* `i+1` *dans le match l'opposant à* `j+1` *et dans* `ch[i][j][1]` *on trouve le nombre de bus obtenu par* `j+1` *dans le match l'opposant à* `i+1`*. On remarque que le match* `ch[i][j]` *(match aller) n'est pas le même que* `ch[j][i]` *(match retour). Finalement, on suppose que lorsqu'une équipe gagne un match, elle obtient 3 points, 1 seul point pour un match nul et 0 point dans le cas où elle perd le match.*

1. *Écrire une fonction* `nombrePoints` *qui prend en arguments une liste championnat* `ch` *de longueur* `n` *et le numéro d'une équipe (entre 1 à* `n`*) et qui renvoie le nombre de point(s) obtenu(s) par cette équipe pendant le championnat.*
2. *Écrire une procédure* `stockerScore` *qui prend en arguments une liste championnat* `ch` *de longueur* `n`*, le numéro* `i` *d'une équipe, le numéro* `j` *d'une autre équipe et le score du match de* `i` *contre* `j` *et qui met à jour la liste* `ch`*.*
3. *Écrire une fonction champion qui prend en argument une liste championnat* `ch` *et qui renvoie le numéro de l'équipe championne. Une équipe est championne si elle a strictement plus de points que toutes les autres. Si plusieurs équipes ont le même nombre de points, alors une équipe est meilleure si elle a marqué strictement plus de buts. Dans le cas d'égalité parfaite (même nombre maximum de points et même nombre maximum de buts marqués), la méthode renverra 0 pour signaler l'impossibilité de désigner un champion.*

□

Exercice 12 (Maxima locaux, ⋆⋆)

Écrire une fonction qui prend en paramètre une liste d'entiers à deux dimensions (rectangulaire) et calcule le **nombre** *d'entrées* **intérieures** *de la matrice dont tous les voisins sont strictement plus petits. Chaque entrée intérieure de la matrice a quatre voisins (à gauche, à droite, vers le haut, vers le bas). Par exemple, pour la matrice*

```
1 4 9 1 4
4 8 1 2 5
4 1 3 4 6
5 0 4 7 6
2 4 9 1 5
```

la fonction devrait renvoyer 2 *car il y a deux éléments de la matrice (*8 *et* 7*) qui ont uniquement des voisins plus petits.* □