

Introduction à la Programmation 1 PYTHON

51AE011F

Séance 1 de cours/TD

Université Paris-Diderot

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.
- Les **instructions** décrivent des actions à effectuer.
- Les **affectations** stockent des données dans des **variables**.

```
1     stopSecs = (stopHour * 60 + stopMin) * 60
2     startSecs = (startHour * 60 + startMin) * 60
3     numberOfSecs = stopSecs - startSecs
4     alertCode = 0
5     for i in range(0, numberOfSecs, 1):
6         time.sleep(1)
7         if (numberOfSecs - i < 30) :
8             alertCode = 1
9         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) deux expressions séparées par une opération arithmétique binaire (`1 + 2`, `1 + 2 * 3 // 4`, ...);
 - (c) une expression entourée de parenthèses (`(1 + 2)`, `(1 + 2 * 3 // 4)`, ...);
 - (d) une expression précédée du signe moins (`-2`, `-(1 + 2)`, ...).
- Les opérateurs ont des priorités relatives, par exemple $\{*, //, \% \} \succeq \{+,-\}$ où \succeq 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 les bornes 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 * 4 + 3`.

Exercice 2 (Python comme une calculatrice, ★)

Prévoir l'évaluation des expressions arithmétiques suivantes :

```
1 6 * 7 + 3
2 6 * (7 + 3)
3 45 // 7
4 3 * 7 // 4
5 (3 * 7) // 4
6 (45 // 7) * 7 + 45 % 7
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.
- 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 affectées 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

- **ATTENTION** : Ne jamais utiliser dans une expression une variable qui n'a pas été affectée auparavant

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.

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

□

4 Fonctions et procédures

Nommer un calcul comme une fonction [COURS]

— La fonction qui attend en paramètre une valeur x et calcule une valeur qui est le triple de x s'écrit :

```
1 def triple (x) :  
2     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.

— 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 :

```
1 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 : on les sépare par des virgules.

— On peut utiliser des fonctions prédéfinies dans des *bibliothèques de fonctions*.

Exercice 7 (Utiliser une fonction, *)

Étant donnée la fonction suivante, que vaut l'expression `twice(3)` ?

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

□

Exercice 8 (Utiliser une fonction, **)

On considère la fonction suivante.

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

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

```
1 b = 5  
2 a = 3  
3 a = b - a  
4 a = cube(a)
```

□

Exercice 9 (Écrire une fonction, **)

Voici une fonction

```
1 def f (x) :  
2     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.

□

Utiliser une procédure _____ [COURS]

- 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 :

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

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

```
1 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`.

Écrire une procédure _____ [COURS]

- 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 :

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

- On remarquera que contrairement aux fonctions, les procédures ne retournent pas de valeurs !

Exercice 11 (Différences syntaxiques, **)

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

```
1 x = 10
2 y = 42
3 if (x <= 0) :
4     y = x
5 else :
6     y = -x
```

pour affecter la valeur de x à y si $x \leq 0$ 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` :
 $e1 == e2$, $e1 != e2$, $e1 < e2$, $e1 \leq e2$, $e1 > e2$, $e1 \geq e2$.
- 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` ?

```
1 x = 3
2 y = 4
3 xy_max = 0
4 if (x > y) :
5     xy_max = x
6 else :
7     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 :

```
1 a = 2
2 b = a * a + 3
3 c = b - a
4 if (c == a) :
5     a = 1
6 else :
7     a = a + 3
8 if (b + c < a) :
9     b = 2
10 else :
11     b = 4
12 if (b != c * c) :
13     c = 12
14 else :
15     c = -6
```

□

6 Boucles

Boucles (bornées) [COURS]

- Une boucle 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 :

```
1 for i in range (0, 10, 1) :  
2     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)`. 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 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.

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

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

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

□

Exercice 19 (Boucles simples, **)

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

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

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

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

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

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

□

Exercice 20 (Des entiers qui s'ajoutent, **)

Quelle est la valeur de `sum` après la suite d'instructions suivante :

```
1 i = 0
2 sum = 0
3 sum = sum + i
4 i = i + 1
5 sum = sum + i
6 i = i + 1
7 sum = sum + i
8 i = i + 1
9 sum = sum + i
10 i = i + 1
11 sum = sum + i
12 i = i + 1
13 sum = sum + i
14 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 10 fois avec `i` prenant pour valeurs les différents entiers entre 1 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 :

```
1 bound = 5
2 sum = 0
3 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

51AE011F

Séance 2 de cours/TD

Université Paris-Diderot

Objectifs:

- Découverte du type `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.

1 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 `""`.

Exercice 1 (De premiers exemples, ☆)

Quelle est la valeur des variables après l'exécution des instructions suivantes ?

```
1 s = " rentre chez lui."
2 s2 = "Paul"
3 s3 = "Michel"
4 s4 = s2 + s
5 s5 = s3 + s
6 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.

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

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

```
1 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".

2 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 :

```
1 def add (x) :  
2     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 affichée 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 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 de caractères numériques, par exemple `int("ab")` génère une erreur.

Exercice 2 (Des exemples simples avec des données de différents type, ☆)

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

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

2. Que fait la suite d'instructions suivantes ?

```
1 x = 3  
2 st = "Ca va ?"  
3 ch = ""  
4 x = x // 2  
5 if (x == 1) :
```

```

6     ch = "How are you ?"
7 else :
8     ch = "Comment allez-vous ?"
9 ch = ch + "\n"
10 print (ch)

```

3. Que fait la suite d'instructions suivantes ?

```

1 s = "Anticonstitutionnellement"
2 a = len (s)
3 if (a % 2 == 0) :
4     print ("La longueur est paire")
5 else :
6     print ("La longueur est impaire")

```

□

Exercice 3 (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 4 (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 considèrera comme type ici que `int` et `str`.

```

1 def f (x) :
2     y = x % 2
3     if (y == 0) :
4         print ("Argument pair")
5     else :
6         print ("Argument impair")
7
8 def g (x) :
9     y = x % 2
10    if (y == 0) :
11        return "Argument pair"
12    else :
13        return "Argument impair"
14
15 def h (x, y) :
16    return (x + y)
17
18 def id (x) :
19    return x

```

code/exolnference.py

□

3 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 :

```
1 def sandwich (s, s2) :
2     return s + s2 + s
3
4 def neymarOrNot (x) :
5     st = ""
6     if (x == 10) :
7         st = "C'est le numéro de Neymar."
8     else :
9         st = "Ce n'est pas le numéro de Neymar."
10    return st
```

Elles peuvent être utilisées de la façon suivante :

```
1 s = sandwich ("Hello", "World!\n")
2 s2 = sandwich (s, "Ca va ?")
3 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.
- **Remarque :** La procédure `print` ajoute systématiquement un retour à la ligne après l'affichage, si l'on veut éviter cela, il faut appeler `print` avec un deuxième paramètre 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="")`.

Exercice 5 (Utilisation de fonctions données, ☆)

1. On considère la fonction suivante :

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

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

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

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

```
1 ch = "Ceci est "
2 ch2 = "une chaîne"
3 ch = ch + ch2
4 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 associé 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 6 (Parcourir une chaîne de caractères, ☆)

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

```
1 ch = "Avec consonnes"  
2 cha = ch[0]  
3 cha = cha + ch[2]  
4 cha = cha + ch[6]  
5 cha = cha + ch[9]  
6 cha = cha + ch[12]
```

□

Exercice 7 (Modification d'une fonction, ☆☆)

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

Et on considère la liste d'instructions suivante :

```
1 a = 4  
2 b = a * a  
3 a = b - a;  
4 s = message (a)  
5 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).

□

4 Expressions booléennes dans les conditionnelles

Des expressions booléennes plus complexes dans les tests _____[COURS]

- Une expression booléenne a le type `bool`.
- Une expression booléenne peut s'évaluer en l'une des deux valeurs `True` ou `False`.
- 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`.
- Exemples d'expressions utilisant ces opérateurs :
 - `(2 < x and x <= 5)`
 - `(x == 1 or x == 2)`
 - `not(x == 0)` (qui est équivalent à `"x != 0"`)
- 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`.
- **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 :

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

Exercice 8 (Savoir évaluer une condition, ★)

Quelle est la valeur de la variable `x` après la suite d'instructions suivante ?

```
1 a = 2
2 a = a * a * a * a + 1
3 b = a // 2
4 c = a - 1
5 x = 0
6 if ((b == 0) and (c == 16)) :
7     x = 1
8 else :
9     x = 2
```

□

Exercice 9 (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 :

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 10 (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)

□

5 Instructions conditionnelles imbriquées

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 :

```
1 x = 1
2 y = 2
3 z = 0
4 if (x == 1) :
5     if (y == 2) :
6         z = 3
7     else :
8         z = 5
9 else :
10    z = 7
```

L'exécution du programme précédent affecte la valeur 3 à la variable z.

- 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.
- 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 :

```
1 x = 3
2 if (x == 0) :
3     print ("x est nul ")
4 else :
5     if (x == 1) :
6         print ("x vaut un ")
7     else :
8         if (x == 2) :
9             print ("x vaut deux")
10        else :
11            print ("x est négatif ou plus grand que deux")
```


peut s'écrire :

```
1 x = 3
2 if (x == 0) :
3     print ("x est nul ")
4 elif (x == 1) :
5     print ("x vaut un ")
6 elif (x == 2) :
7     print ("x vaut deux")
8 else :
9     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 :

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

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

Exercice 11 (Utilisation des opérateurs booléens, **)

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

1.

```
1 def f (l) :
2     x = 0
3     if (l != 0) :
4         if ( l <= 10) :
5             x = 1
6         else :
7             x = 2
8     else :
9         x = 2
10    return x
```

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

2.

```
1 def g (a) :
2     b = 2 * a
3     b = a - a
4     if (b == a) :
5         b = 0
6     else :
7         if (b > 43) :
8             b = 0
9         else
10            b = 1
11    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.

```
1 def h (c, d) :
2     e = 0
3     if (d == 6) :
4         e = 3
5     else :
6         if (c >= 2) :
7             e = 2
8         else
9             e = 3
10    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.

□

6 Boucles

Exercice 12 (Pendur à l'envers, **)

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

Contrat:

`paramètre = "amphitheatre" → affichage : -----`

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

Contrat:

`paramètre = "amphitheatre" → affichage : ertaehtihpma`

□

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.

```
1 for i in range (0, 100, 1) :
2     if (i % 2 == 0) :
3         print ("Pair.")
4     else :
5         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 13 (Effacement des espaces, *)

Écrire une fonction 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

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 14 (Concaténation, *)

Écrire une fonction 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 15 (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 16 (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 17 (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 18 (Somme des cubes, *)

Écrire une boucle permettant de calculer la somme des cubes des entiers de 1 à 100.

□

Exercice 19 (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 fonction `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 20 (Simplification de code, ★)

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

```

1 def f (x, y) :
2     if (((x <= y) or (x >= y + 1))) :
3         if ((x <= y) and (x < y)) :
4             if (x < -x) :
5                 a = -x
6                 return a
7             else :
8                 return x
9         else :
10            if (y * y * y < 0) :
11                return -y
12            return y
  
```

2. Écrire la même fonction de manière plus simple.

□

Exercice 21 (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; assez bien entre 12 inclus et 14 exclu; bien entre 14 inclus et 16 exclu, et très bien au-delà de 16.

□

Exercice 22 (Lignes et pointillés, ★★)

1. Écrivez une fonction 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 fonction pour qu'elle affiche une ligne pointillée alternant astérisques et espaces. Par exemple, si n vaut 7, votre fonction affichera :

* * * *

Qu'affiche votre fonction si n est pair ?

□

Exercice 23 (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 fonction *divisors* qui prend en paramètre un entier n et affiche tous les entiers strictement inférieurs à n qui divisent n .

Contrat:

$n = 1 \rightarrow$ affichage :
 $n = 5 \rightarrow$ affichage : 1
 $n = 60 \rightarrow$ 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 $n \leq m$, " n est parfait" (en remplaçant avec la valeur de n).

□

Exercice 24 (Occurrences d'un caractère, **)

Écrire une fonction *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:

$(s,c) = (\text{"abracadabra"}, \text{"a"}) \rightarrow$ affichage : 0 3 5 7 10

□

Introduction à la Programmation 1 PYTHON

51AE011F

Séance 3 de cours/TD

Université Paris-Diderot

Objectifs:

- Manipuler des boucles avec accumulateurs
- Manipuler deux boucles imbriquées

1 Accumuler des valeurs grâce aux boucles

Utilisation d'accumulateur 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 fragment de code suivant calcule la somme des carrés des entiers de 1 à 100 :

```
1 s = 0
2 for i in range (1, 101, 1) :
3     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, ★)

```
1 st = "";
2 for i in range (1, 51, 1) :
3     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 "aaaaa ... 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'instruction `input ()` permet de lire une chaîne de caractères tapée au clavier et de la retourner. Si on veut lire un entier tapé au clavier, il suffit de convertir cette chaîne en entier en faisant `int (input ())`.

Écrire un fragment de code 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 :

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

affiche :

```
1 i = 0, j = 1  
2 i = 0, j = 2  
3 i = 0, j = 3  
4 i = 0, j = 4  
5 i = 1, j = 1  
6 i = 1, j = 2  
7 i = 1, j = 3  
8 i = 1, j = 4  
9 i = 2, j = 1  
10 i = 2, j = 2  
11 i = 2, j = 3  
12 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 fragment de code 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 fonction 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, ★)

Écrire une fonction "power (x , n)" qui renvoie la valeur x^n .

□

Exercice 7 (Factorielle, ★)

Écrire une fonction "fact (n)" qui renvoie la factorielle de n .

□

Exercice 8 (Ça use, ★★)

1. Écrire un fragment de code 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, ★★)

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, ★★)

1. Écrire un fragment de code 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, ★★)

1. Écrire une fonction 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 fonction pour qu'elle affiche un carré creux, par exemple pour n valant 4,

* *

* *

□

Exercice 12 (Séries, * - *)**

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

□

Exercice 13 (Sous-chaîne, **)

Écrire une procédure *sousChaîne* 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:

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

□

Introduction à la Programmation 1 PYTHON

51AE011F

Séance 4 de cours/TD

Université Paris-Diderot

Objectifs:

- Contrôle continu (45 min, 3 exercices tirés des travaux dirigés)
- 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 dans comme deuxième valeur et 12 comme troisième valeur sera notée `[2, 4, 12]`.
- 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 de valeurs 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 égales 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, ☆)

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 (Lecture, ☆)

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

```
1 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 :

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

□

Exercice 3 (Swap, ☆)

On considère la déclaration suivante :

```
1 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 Opération 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]`.

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

Exercice 4 (Création de listes, ☆)

1. Donner les instructions pour créer une liste `a` valant `{1, 2, 3, 4, 5, ..., 1000}`.
(Il est bien sûr recommandé d'utiliser une boucle `for` pour remplir une telle liste.)

□

Exercice 5 (Manipulation de listes, ☆)

Donner les instructions réalisant les opérations suivantes :

1. Créer une liste `l1` valant $\{2, 4, 6, 8, 10, 12, 14, 16\}$.
2. Créer une liste `l2` de la même taille que `l1` qui contient à chaque indice le double de la valeur contenue dans `l1` à l'indice correspondant (même si l'on peut créer `l2` manuellement, l'idée ici est de remplir la liste `l2` en même temps que l'on parcourt la liste `l1` à l'aide d'une boucle `for`).

□

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 (Liste d'éléments non nuls, ★★)

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

```

1 def notZero (l) :
2   n = len (l)
3   s = 0
4   for i in range (0, n, 1) :
5     if (l[i] != 0) :
6       s = s + 1
7   return s

```

2. Donner une séquence d'instructions réalisant les actions suivantes :

- (a) Création d'une liste `l1` valant : $[4, 3, 6, 2, 0, 0, 2, 0, 4]$.
- (b) Création d'une liste `l2` contenant les éléments de la liste `l1` qui sont différents de zéro (dans le même ordre). Pour cela vous utiliserez la fonction `notZero`.

□

3 Do it yourself

Exercice 8 (Somme, ★)

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

□

Exercice 9 (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 10 (Grognements, ★)

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

□

Exercice 11 (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 12 (Première occurrence, **)

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 13 (Pyramide, ***)

Écrire une fonction 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 14 (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 15 (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 $[l1[0], l2[0], l1[1], l2[1], \dots, l1[n], l2[n]]$ (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 16 (Plagiat, **)

- 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 .
- Que se passe-t-il si on appelle la fonction `plagiarism` sur la même liste `plagiarism (l, s)` ?
- 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 17 (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. \square

Exercice 18 (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 placées, 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 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 placées et de mal placées. 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 19 (Réordonnancement, **)

Écrire une fonction qui demande un entier n à l'utilisateur, puis lui demande n fois d'entrer un entier x ainsi qu'une position $j \in \{1, \dots, n\}$ (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") □