Formatage des affichages

Dans tout langage, il est essentielle de pouvoir formater les sorties, c'est-à-dire de pouvoir décider et fixer comment les nombres sont affichés (que ce soit dans un fichier ou dans le terminal).

Formater signifie donner un format. Cela a plusieurs intérêts :

- écrire plus joliement en permettant par exemple d'aligner correctement les nombres ⇒ cela facilite la lecture ;
- avoir un format identique et prévisible pour les grands tableaux de nombres écrits dans des fichiers ⇒ cela facilite l'automatisation.

Le formatage consiste à donner une règle à `python` pour transformer les données à afficher en une chaîne de caractères. Plusieurs possibilités, plus ou moins anciennes, existent : nous utiliserons la méthode la plus moderne et la plus efficace, les `fstring` .

Une `fstring` est une chaîne de caractères précédée du caractère `f` . Par exemple `f"toto"` . Il est alors possible d'afficher une `fstring` à l'aide de la commande `print` (nous verrons plus loin comment remplacer `print` pour écrire directement dans un fichier).

In [1]:
```
ma_fstring = f"toto"
print(ma_fstring)
```

```
toto
```

L'intérêt des `fstring` est qu'il est possible d'affichier des variables facilement en les entourant par `{` et `}` . Voici un exemple

In [2]:
```
x = 10
print(f"La valeur de 3*x est {3*x}")
```

```
La valeur de 3*x est 30
```

Il est alors possible de formater l'affichage de la variable `x` en faisant suivre dans les accolades la valeur à afficher par `:` puis par le format choisi. Par exemple, pour afficher un nombre entier `int` , on peut choisir de l'afficher en base 10 (`d`), en base 8 (`o`), en base 2 (`b`) ou en base 16 (`x`). On fait également précédé ce symbole du nombre 5 pour imposer la réservation de 5 cases pour l'affichage.

In [8]:
```
print("-"*33)
print("|   d   |   b   |   o   |   x   |")
print("-"*33)
for n in range(25):
    print(f"| {n:5d} | {n:5b} | {n:5o} | {n:5x} |")
print("-"*33)
```

```
---------------------------------
|   d   |   b   |   o   |   x   |
---------------------------------
|     0 |     0 |     0 |     0 |
|     1 |     1 |     1 |     1 |
|     2 |    10 |     2 |     2 |
|     3 |    11 |     3 |     3 |
|     4 |   100 |     4 |     4 |
|     5 |   101 |     5 |     5 |
|     6 |   110 |     6 |     6 |
|     7 |   111 |     7 |     7 |
|     8 |  1000 |    10 |     8 |
|     9 |  1001 |    11 |     9 |
|    10 |  1010 |    12 |     a |
|    11 |  1011 |    13 |     b |
|    12 |  1100 |    14 |     c |
|    13 |  1101 |    15 |     d |
|    14 |  1110 |    16 |     e |
|    15 |  1111 |    17 |     f |
|    16 | 10000 |    20 |    10 |
|    17 | 10001 |    21 |    11 |
|    18 | 10010 |    22 |    12 |
|    19 | 10011 |    23 |    13 |
|    20 | 10100 |    24 |    14 |
|    21 | 10101 |    25 |    15 |
|    22 | 10110 |    26 |    16 |
|    23 | 10111 |    27 |    17 |
|    24 | 11000 |    30 |    18 |
---------------------------------
```

Il est également possible de remplir les espaces avec des zéros et d'aligner les nombres. Voici des exemples que vous pouvez tester.

In [9]:
```
print("-"*(10*4+1))
print("|    d    |    b    |    o    |    x    |")
print("-"*(10*4+1))
for n in range(25):
    print(f"| {n:^7d} | {n:>7b} | {n:07o} | {n:07x} |")
print("-"*(10*4+1))
```

```
----------------------------------------
|    d    |    b    |    o    |    x    |
----------------------------------------
|    0    |        0 | 0000000 | 0000000 |
|    1    |        1 | 0000001 | 0000001 |
|    2    |       10 | 0000002 | 0000002 |
|    3    |       11 | 0000003 | 0000003 |
|    4    |      100 | 0000004 | 0000004 |
|    5    |      101 | 0000005 | 0000005 |
|    6    |      110 | 0000006 | 0000006 |
|    7    |      111 | 0000007 | 0000007 |
|    8    |     1000 | 0000010 | 0000008 |
|    9    |     1001 | 0000011 | 0000009 |
|   10    |     1010 | 0000012 | 000000a |
|   11    |     1011 | 0000013 | 000000b |
|   12    |     1100 | 0000014 | 000000c |
|   13    |     1101 | 0000015 | 000000d |
|   14    |     1110 | 0000016 | 000000e |
|   15    |     1111 | 0000017 | 000000f |
|   16    |    10000 | 0000020 | 0000010 |
|   17    |    10001 | 0000021 | 0000011 |
|   18    |    10010 | 0000022 | 0000012 |
|   19    |    10011 | 0000023 | 0000013 |
|   20    |    10100 | 0000024 | 0000014 |
|   21    |    10101 | 0000025 | 0000015 |
|   22    |    10110 | 0000026 | 0000016 |
|   23    |    10111 | 0000027 | 0000017 |
|   24    |    11000 | 0000030 | 0000018 |
----------------------------------------
```

Il est possible d'automatiser tout cela en imbriquant les `fstring` :

In [3]:
```
# format
# N : le nombre de cases à réserver
# d : pour avoir un entier écrit en écriture décimale
# {variable:Nd}

x = 5
print("-"*((x+3)*4+1))
print(f"| {'d':^{x}} | {'b':^{x}} | {'o':^{x}} | {'x':^{x}} |")
print("-"*((x+3)*4+1))
for n in range(25):
    print(f"| {n:{x}d} | {n:{x}b} | {n:{x}o} | {n:{x}x} |")
print("-"*((x+3)*4+1))
```

```
---------------------------------
|   d   |   b   |   o   |   x   |
---------------------------------
|     0 |     0 |     0 |     0 |
|     1 |     1 |     1 |     1 |
|     2 |    10 |     2 |     2 |
|     3 |    11 |     3 |     3 |
|     4 |   100 |     4 |     4 |
|     5 |   101 |     5 |     5 |
|     6 |   110 |     6 |     6 |
|     7 |   111 |     7 |     7 |
|     8 |  1000 |    10 |     8 |
|     9 |  1001 |    11 |     9 |
|    10 |  1010 |    12 |     a |
|    11 |  1011 |    13 |     b |
|    12 |  1100 |    14 |     c |
|    13 |  1101 |    15 |     d |
|    14 |  1110 |    16 |     e |
|    15 |  1111 |    17 |     f |
|    16 | 10000 |    20 |    10 |
|    17 | 10001 |    21 |    11 |
|    18 | 10010 |    22 |    12 |
|    19 | 10011 |    23 |    13 |
|    20 | 10100 |    24 |    14 |
|    21 | 10101 |    25 |    15 |
|    22 | 10110 |    26 |    16 |
|    23 | 10111 |    27 |    17 |
|    24 | 11000 |    30 |    18 |
---------------------------------
```

Les formats possibles sont

- `d` : entier en base 10 signé
- `o` : entier en base 8 non signé
- `x` : entier en base 16 non signé (lettres en minuscule)
- `X` : entier en base 16 non signé (lettres en majuscule)
- `e` : réel en format exponentiel (e en minuscule)
- `E` : réel en format exponentiel (E en majuscule)
- `f` ou `F` : réel en format nombre à virgule
- `g` : choisi entre `e` et `f` selon l'exposant du nombre
- `G` : choisi entre `E` et `F` selon l'exposant du nombre
- `c` : un unique caractère qui peut être un entier ou un caractère
- `r` : chaine de caractère (en utilisant la fonction membre `repr()`)
- `s` : chaine de caractère (en utilisant la fonction membre `str()`)

NB : pour les formats qui permettent d'afficher des nombres à virgules (`F` , `E` , `G`), il est possible d'imposer à la fois la place réservée et le nombre de chiffre après la virgule. Par exemple, le format `10.7F` réserve 10 cases avec 7 chiffres après la virgule pour un nombre flottant.

$$\pm\star,\star\star\star\star\star\star\star$$

Autre exemple, le format `10.3E` réserve 10 cases avec 3 chiffres après la virgule pour un nombre au format exponentiel.

$$\pm\star,\star\star\star E\pm\star\star$$

In [104]:
```
n = 3
x = 3.14
print(f"L'entier n vaut {n:d}.")
print(f"L'entier n vaut {n:2d}.")
print(f"L'entier n vaut {n:02d}.")
print(f"Le réel x vaut {x:f}.")
print(f"Le réel x vaut {x:4.2f}.")
print(f"Le réel x vaut {x:10.7f}.")
print(f"Le réel 1000x vaut {1000*x:10.3e}.")
print(f"Le réel .001x vaut {.001*x:10.3e}.")
```

```
L'entier n vaut 3.
L'entier n vaut  3.
L'entier n vaut 03.
Le réel x vaut 3.140000.
Le réel x vaut 3.14.
Le réel x vaut  3.1400000.
Le réel 1000x vaut  3.140e+03.
Le réel .001x vaut  3.140e-03.
```

Il est possible d'ajouter des caractères spéciaux comme `\n` pour aller à la ligne ou `\t` pour faire une tabulation.

In [107]:
```
print(f"L'entier n vaut \t{n:02d}.\nLe réel x vaut \t\t{x:4.2f}.")
```

```
L'entier n vaut 	03.
Le réel x vaut 		3.14.
```

In [12]:
```
import math

x = -math.sqrt(2)
print(f"x = {x:12.1F}|")
print(f"x = {x:12.9F}|")
print(f"x = {x:12.5E}|")
```

```
x =         -1.4|
x = -1.414213562|
x = -1.41421E+00|
```

In [ ]:
```

```