2 Dictionnaires

Le problème d'une liste c'est que l'accès à un de ses éléments nécessite de connaître son indice. Les dictionnaires permettent un accès plus souple aux éléments qu'ils contiennent.

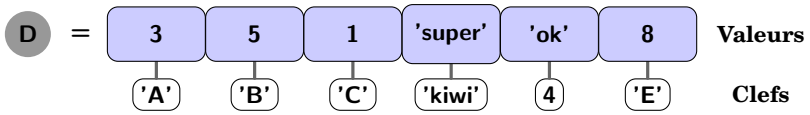
Définition
Un **dictionnaire** `D` est constitué de valeurs `v` accessibles avec une clef `k` en effectuant `D[k] = v`.
Exemple
Considérons le dictionnaire `D` défini par :

Clef	'A'	'B'	'C'	'fred'	4	'E'
Valeur	3	5	1	'super'	'ok'	8

- L'instruction python permettant d'obtenir `D` est :

```
D = {'A': 3, 'B': 5, 'C': 1, 'fred':'super', 4 :'ok', 'E':8}
```

- Voici une représentation de `D` voisine de celle d'une liste :



Actions possibles

Instruction	Résultat
Créer un dictionnaire vide `D`	
`D = {}`	
Obtenir la valeur `value` associée à une clef `key` d'un dictionnaire `D`	
`D[key]`	`value`
Soit le dictionnaire `D = {'A': 3, 'B': 2, 'F': '0'}`	
`D['A']`	`3`
Obtenir la liste des clefs de `D`	
`D.keys()`	`dict_keys(['A', 'B', 'F'])`
Obtenir la liste des valeurs de `D`	
`D.values()`	`dict_values([3, 2, '0'])`
Obtenir le dictionnaire `D` d'éléments `k : 0`, avec $k \in [\![0,3]\!]$	
`{k :0 for k in range(4)}`	`{0: 0, 1:0, 2:0, 3:0}`

D Modules et fonctions

Qu'est-ce qu'un module ?
On appelle **module** tout fichier constitué de code **Python** (c'est-à-dire tout fichier avec l'extension **. py**) importé dans un autre fichier ou script.
Pour notre usage, les principaux modules utilisés seront **math**, **pylab**, **random**, **pandas**, **geopandas**, **plotly**, **ipywidgets**, **seaborn**, **networkx**, **scipy.stats**, **sympy**, **numpy**.

1 Module math

Le module **math** contient un certain nombre de commandes utiles pour faire des calculs.
En voici quelques unes :

Objectif	Avec Python
Chargement intégral du module **math**	`from math import *`
Calculer $\lfloor x \rfloor$ et $\sqrt{x}$	`floor(x)` et `sqrt(x)`
Calculer $\cos(x)$ et $\sin(x)$	`cos(x)` et `sin(x)`

Objectif	Avec Python
Calculer $\ln(x)$ et $\mathrm{Log}(x)$	`log(x)` et `log(x,10)`
Calculer e^x	`exp(x)`
Écrire π et $\mathrm{e} = \mathrm{e}^1$	`pi` et `exp(1)`

2 Module pylab

Le module **pylab** permet de représenter une liste en fonction d'une autre, pour construire une courbe, etc.

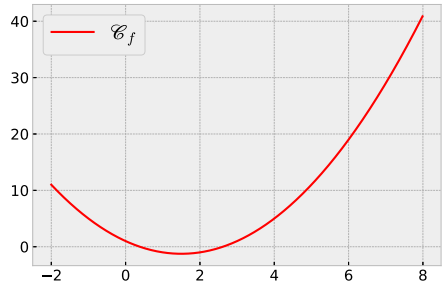
Objectif	Avec Python
Chargement du module **pylab** sous le pseudo **pl**	`import pylab as pl`
Point de coordonnées $(2\,;3)$	`pl.plot(2, 3, 'o')`
Points de coordonnées $(2\,;3)$ et $(5\,;6)$	`pl.plot([2, 5], [3, 6], 'o')`
Nuage de points de coordonnées $(x_i\,;y_i)$ pour chaque valeur de i avec $1 \leqslant i \leqslant r$, lorsque • $X = [x_1, x_2, \ldots, x_r]$ • $Y = [y_1, y_2, \ldots, y_r]$.	`pl.plot(X, Y, 'o')`
Diagramme en barres avec • en abscisses $X = [x_1, x_2, \ldots, x_r]$ • de hauteurs $Y = [y_1, y_2, \ldots, y_r]$.	`pl.bar(X, Y, color = 'red', alpha = 0.5)` Le paramètre `alpha` permet de gérer l'opacité de la couleur des barres.
Droite d'équation $x = a$, en gris	`pl.axvline(x = a, color = 'gray')`
Droite d'équation $y = a$, en gris	`pl.axhline(y = a, color = 'gray')`
Gestion taille de la fenêtre $x_{\min}$, $x_{\max}$, $y_{\min}$ et $y_{\max}$	`pl.axis([xmin, xmax, ymin, ymax])`

Voici quelques exemples :

1. Chargeons le module pylab et indiquons le style utilisé (en bonus) :

```
In [1]: import pylab as pl
        pl.style.use('bmh')
```

Représentons la courbe représentative de f définie par $f(x) = x^2 - 3x + 1$ sur $[-2, 8]$:

```
In [2]: X = [-2 + k / 100 for k in range(1000)]
        Y = [x ** 2 - 3 * x + 1 for x in X]
        pl.plot(X, Y, color = 'red',
                label = r'$\mathscr{C}_f$')
        pl.legend(fontsize = 18)
```

Out[2]: