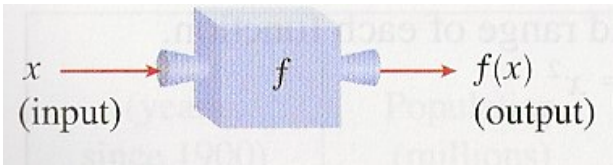
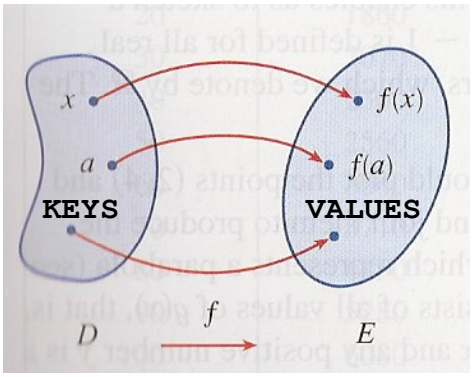
Cours 9 – Les dictionnaires

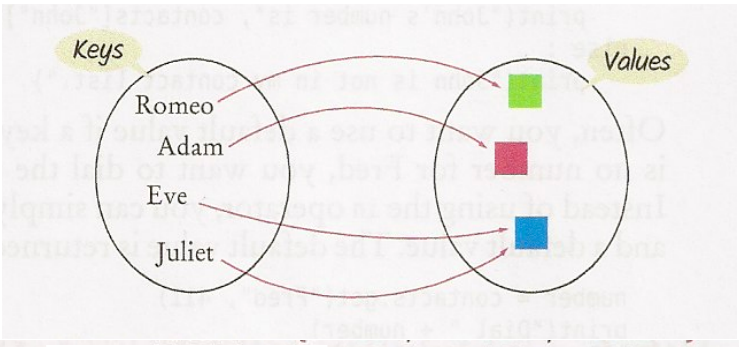
Pour stocker des données, nous avons vu jusqu'à présent trois types de **conteneur** ou **itérables** : les chaînes de caractères (`str`), les listes (`list`) et les tableaux (`array`). Il existe d'autres façons dans Python de stocker des données. Nous allons découvrir à présent les dictionnaires (`dictionary` (ies) pluriel).
Comme les listes, **les dictionnaires sont des itérables modifiables**.

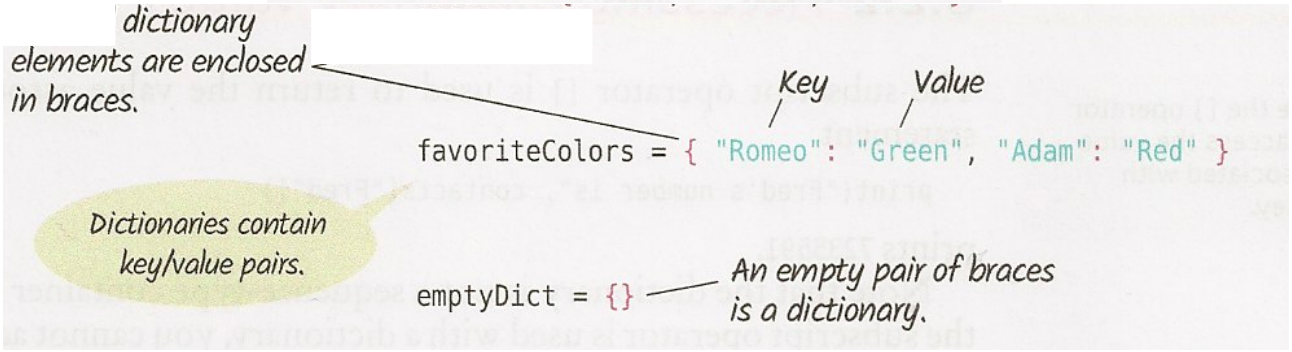
En mathématiques, une application $f:\begin{cases} D\subset\Re\to E\subset\Re \\ x\mapsto y \end{cases}$ est une « règle » qui associe à chaque $x\in D$ (D est ensemble fini) exactement un seul élément $y=f\left(x\right)\in E$ (E est ensemble fini).





f est une application surjective (cf. cours de Mathématiques) si chaque valeur de E à au moins une antécédent.
De façon similaire **le dictionnaire est une « application » surjective :** il associe à une clé unique (`key`) une valeur (`value`). Une **valeur** donnée peut-être associée à plusieurs clés ainsi `len(E)<=len(D)` **.**





ATTENTION : Contrairement aux éléments d'une liste, les paires (clé, valeur) d'un dictionnaire ne sont pas stockées dans un ordre particulier et ne peuvent donc pas être accessibles par leur position. La commande ~~`nom_dictionnaire[i]`~~ n'a pas de sens pour un dictionnaire. Les dictionnaires ne sont pas des séquences comme les listes.

Le tableau ci-dessous donne les principales méthodes et fonctions sur les dictionnaires.

	Synthaxe Python
Dictionnaire	`dict = {cle1:valeur1, cle2:valeur2,…}`
Accès à la valeur d'une clé	`dict[cle]`
Séquence des clés	`dict.keys()`
Séquence des valeurs	`dict.values()`
Séquence des (clé, valeur)	`dict.items()`
Supprimer une paire (clé, valeur)	`dict.pop[cle]` ou `del dict[cle]`
Nombre de clés	`len(dict)`
Copie profonde (idem liste)	`dict_copie=dict.copy()`

A / Création d'un dictionnaire

Le dictionnaire ci-dessous recense les cinq femmes lauréates du prix Nobel de Physique depuis sa création.

```
Nobel_physique_femme={"Curie":1903,"Goeppert-Mayer":1963,"Strickland":2018,"Ghez":2020,"Huillier":2023}
```

La **clé**, ici de type chaîne de caractère correspond au nom de la récipiendaire et l'année de la récompense correspond à la **valeur**, ici un entier.

✓ **Les clés** d'un dictionnaire ne peuvent pas être des listes (`list`), des dictionnaires (`dictionary`) ou des tableaux (`array`). Le seul objet itérable (c'est-à-dire dans lequel on puisse faire une boucle `for` : `for element in objet`) qui puisse être la clé d'un dictionnaire est une chaîne de caractère (`str`) . Par contre la clé peut être un réel (`float`) ou un entier (`int`).
✓ **Une valeur** attachée à une clé peut être de n'importe quel type.

La commande `dict` permet de dupliquer un dictionnaire :

```
New_Nobel_physique_femme=dict(Nobel_physique_femme)
```

Notes : Pour l'histoire, il y a eu huit femmes prix Nobel de Chimie dont trois Françaises : Marie Curie en 1911, sa fille Irène Curie en 1935 et Emmanuelle Charpentier en 2020. Seules six personnes ont obtenu deux prix Nobel, dont une seule femme : Marie Curie. La Canadienne Donna Strickland, pionnière dans le domaine des Lasers, a partagé le prix Nobel de Physique avec Gérard Mourou, son directeur de thèse.

B / Accès aux valeurs d'un dictionnaire

L'opérateur `[]` est utilisé **pour retourner la valeur associée à une clé**. On ne peut pas accéder à une valeur par un indice mais seulement par une clé.

```
print("La physicienne Strickland a eu le prix Nobel de Physique en",Nobel_physique_femme["Strickland"])
```

```
La physicienne Strickland a eu le prix Nobel de Physique en 2018
```

Pour vérifier si une clé est présente dans le dictionnaire, on peut utiliser les opérateurs `in` ou `not in`:

```
if "Isild" in Nobel_physique_femme:
    print("Isild a eu le prix nobel en",Nobel_physique_femme["Isild"] )
else:
    print("Isild n'a pas encore obtenu le prix Nobel de Physique mais c'est pour bientôt :-)")
```

```
Isild n'a pas encore obtenu le prix Nobel de Physique mais c'est pour bientôt :-)
```

C / Manipuler un dictionnaire

C. 1 – Ajout de paires (clé, valeur)

```
Nobel_physique_femme["Gabrielle"]=2040
Nobel_physique_femme["Blanche"]=2041
Nobel_physique_femme["Faustine"]=2042
Nobel_physique_femme["Zoé"]=2043
Nobel_physique_femme["Louise"]=2044
Nobel_physique_femme["Isild"]=2045
Nobel_physique_femme["Emeline"]=2046

if "Zoé" in Nobel_physique_femme:
    print("Zoé a eu le prix nobel en",Nobel_physique_femme["Zoé"] )
else:
    print("Zoé n'a pas encore obtenu le prix Nobel de Physique mais c'est pour bientôt :-)")
```

```
Zoé a eu le prix nobel en 2043
```

C. 2 – Changer la valeur associée à une clé

```
Nobel_physique_femme["Zoé"]=2053
```

```
Zoé a eu le prix Nobel en 2053
```

C. 3 – Enlever un paire (clé, valeur) avec la méthode `.pop()` et la fonction `del`

```
Nobel_physique_femme.pop("Zoé")

if "Zoé" in Nobel_physique_femme:
    print("Zoé a eu le prix Nobel en",Nobel_physique_femme["Soline"] )
else:
    print("Zoé n'a pas encore obtenu le prix Nobel de Physique mais c'est pour bientôt :-)")

print()
print(Nobel_physique_femme)
```

```
Zoé n'a pas encore obtenu le prix Nobel de Physique mais c'est pour bientôt :-)

{'Curie': 1903, 'Goeppert-Mayer': 1963, 'Strickland': 2018, 'Ghez': 2020, 'Huillier': 2023, 'Gabrielle': 2040, 'Blanche': 2041,
'Faustine': 2042, 'Louise': 2044, 'Isild': 2045, 'Emeline': 2046}
```

La méthode `.pop` **élimine à la fois la clé et sa valeur du dictionnaire**. On peut aussi utiliser la fonction `del`.

```
del Nobel_physique_femme["Zoé"]
```

C. 4 – Parcourir un dictionnaire

```
Nobel_physique_femme={"Curie":1903,"Goeppert-Mayer":1963,"Strickland":2018,"Ghez":2020,"Huillier":2023}

print("Dates des prix Nobel de Physique féminins:")

for cle in Nobel_physique_femme:
     print(cle,Nobel_physique_femme[cle])
```

```
Dates des prix Nobel de Physique féminins:
Curie 1903
Goeppert-Mayer 1963
Strickland 2018
Ghez 2020
Huillier 2023
```

On peut vouloir parcourir le dictionnaire sur les **valeurs** plutôt que sur les **clés** en utilisant la méthode `.values`. Cela peut être intéressant pour créer une liste avec uniquement les valeurs des dates du dictionnaire dans notre exemple.

```
Date_Nobel_physique_femme=[] #creation d'une liste vide

for date in Nobel_physique_femme.values():
    Date_Nobel_physique_femme.append(date)

print(Date_Nobel_physique_femme)
```

```
[1903, 1963, 2018, 2020, 2023]
```

D / La méthode `.keys()` et la méthode `.items()`

La méthode `keys()` renvoie la séquence des clés utilisées dans le dictionnaire. Cette séquence peut-être utilisée telle quelle dans les expressions ou convertie en liste si nécessaire avec la fonction intégrée **`list()`**.

```
print(Nobel_physique_femme.keys())

liste_Nobel=list(Nobel_physique_femme.keys())
print(type(liste_Nobel))
print(liste_Nobel)
```

```
dict_keys(['Curie', 'Goeppert-Mayer', 'Strickland', 'Ghez', 'Huillier'])
<class 'list'>
['Curie', 'Goeppert-Mayer', 'Strickland', 'Ghez', 'Huillier']
```

✓ Les clés d'un dictionnaire **n'ont pas d'ordre** (c'est important, nous en reparlerons plus tard). Il n'y a pas de 1ère clé, 2ème clé, etc. Ainsi, les commandes `clefs[0]` ; `clefs[-1]` ; `clefs[:]` ... renvoient toutes la même erreur `'dict_keys' object is not subscriptable`

✓ On ne peut pas copier, ou supprimer un élément d'un tel objet : `clefs.copy()` ; **`del`** `clefs[0]` donnent une erreur `'dict_keys' object doesn't support copy/item deletion`

✓ Et le plus important à noter est que l'objet `dict_keys` reste lié au dictionnaire : il est automatiquement mis à jour dès qu'on modifie le dictionnaire !

On peut itérer dans les clés d'un dictionnaire (**`for`** `clé` **`in`** `dico.keys()`), à la seule condition qu'il ne faut surtout pas modifier le dictionnaire au cours des itérations (ne pas ajouter de nouvelle clé).

```
for k in Nobel_physique_femme.keys():
    print('clé:',k,'--- valeur:',Nobel_physique_femme[k])
```

```
clé: Curie --- valeur: 1903
clé: Goeppert-Mayer --- valeur: 1963
clé: Strickland --- valeur: 2018
clé: Ghez --- valeur: 2020
clé: Huillier --- valeur: 2023
```

La méthode `items()` renvoie la séquence des paires (clé, valeurs) d'un dictionnaire. Ce qui a été dit précédemment sur la méthode `keys()` reste valable pour la méthode `items()`.

```
print(Nobel_physique_femme.items())
```

```
dict_items([('Curie', 1903), ('Goeppert-Mayer', 1963), ('Strickland', 2018), ('Ghez', 2020), ('Huillier', 2023)])
```

E / Exemple : les polynômes en tant que dictionnaire

Il est facile et commode de stocker et de manipuler un polynôme à l'aide d'un dictionnaire. Considérons par exemple le polynôme suivant :

$$P(x) = -1 + x^2 + 3x^7 .$$

On peut utiliser le dictionnaire suivant pour relier **les clés** → **les puissances, avec les valeurs** → **les coefficients** :

```
P={0:-1, 2:1, 7:3}
```

Il est possible d'utiliser une liste mais dans ce cas il faut noter tous les zéros puisque les indices de la liste sont reliés à la puissance :

```
P=[-1,0,1,0,0,0,0,3]
```

L'avantage du dictionnaire est évidente ici, il suffit de stocker uniquement les coefficients non nuls. Pour le polynôme $1+x^{100}$, le dictionnaire nécessite deux valeurs contre 101 éléments avec une liste !!

La fonction suivante permet d'évaluer le polynôme, représenté par un dictionnaire, pour différentes valeurs de x.

```
def eval_poly_dic(poly,x):
    sum=0.0
    for power in poly.keys() :
        sum += poly[power]*x**power
    return sum

P={0:-1, 2:1, 7:3}
x=4
evaluation= eval_poly_dic(P,x)
print("P(",x,")=",evaluation)
```

```
P( 4 )= 49167.0
```

Avec un dictionnaire, il est facile de représenter un polynôme avec des puissances négatives tel que :

$$P(x)=\frac{1}{2}x^{-3}+2x^{4}$$

Cela est plus délicat avec des listes.

```
P={-3:0.5, 4:2}
```

Si l'on essaie de taper pour ce dernier dictionnaire `P[1]`, nous allons obtenir un message d'erreur `KeyError` puisque 1 n'est pas une clé pour `P`. Pour éviter cela, nous pouvons utiliser la méthode `dict.get(clé,valeur)` qui retourne :

- ***clé :*** le nom de la clé de l'élément dont vous souhaitez renvoyer la valeur
- ***valeur :*** (facultatif) valeur à renvoyer si la clé n'est pas trouvée. La valeur par défaut est None.

```
valeur=P.get(5, "not found")
print(valeur)
```

```
not found
```

Dans cet exemple, la valeur retournée est « `not found` » puisque la clé 5 n'existe pas dans le dictionnaire.