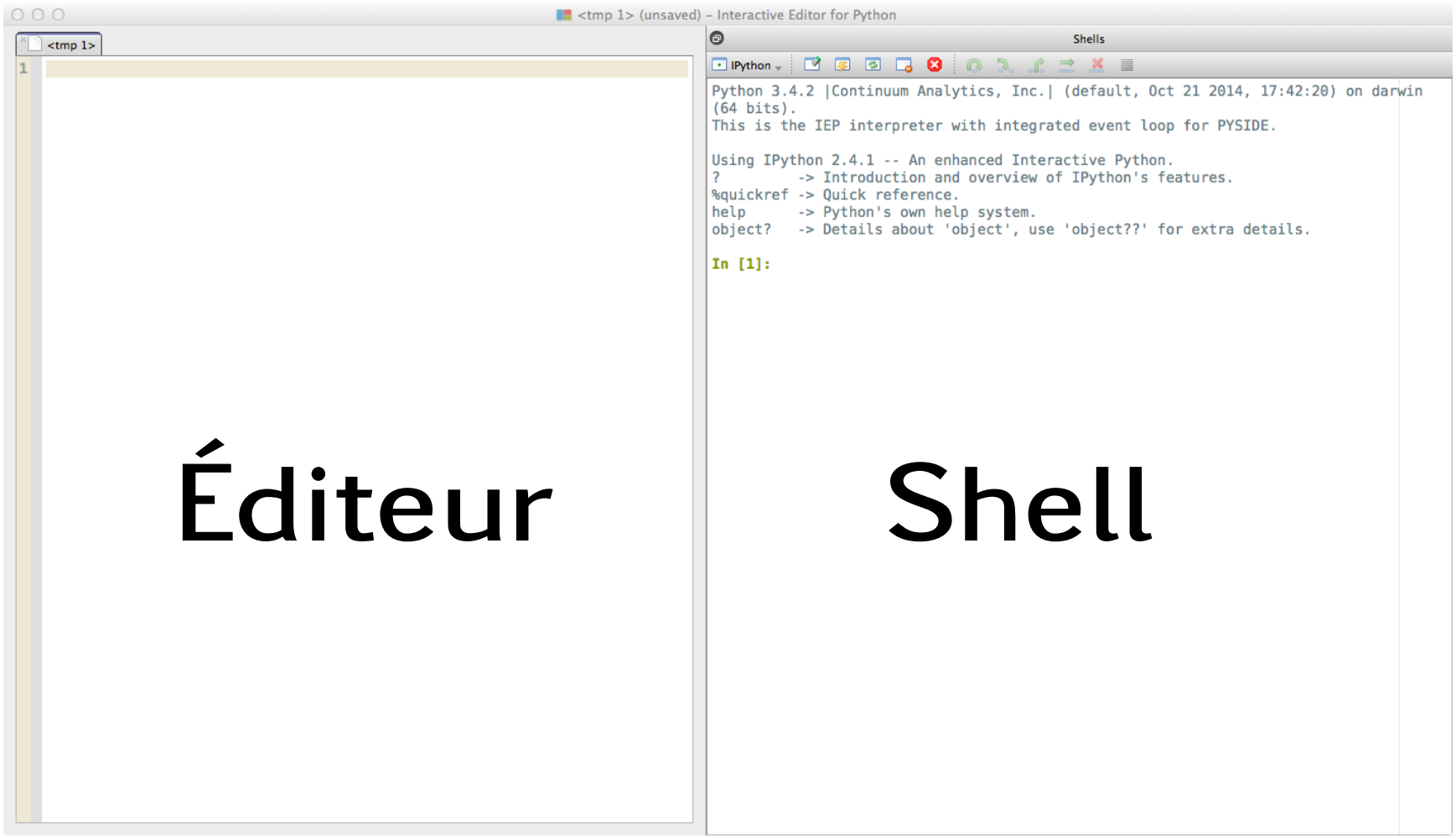
Introduction à PYTHON

Jean-Pierre Becirspahic
Lycée Louis-Le-Grand

Environnement de travail

Dans toute distribution PYTHON il convient de distinguer l'éditeur de l'interface système (le *shell*) :



L'interprète de commande

Différence entre retour et effet

Un retour est le résultat d'un calcul :

```
In [1]: 1 + 1
Out[1]: 2
```

L'interprète de commande

Différence entre retour et effet

Un retour est le résultat d'un calcul :

```
In [1]: 1 + 1
Out[1]: 2
```

Un effet modifie l'environnement :

```
In [2]: print('Hello world !')
Hello world !
```

L'interprète de commande

Différence entre retour et effet

Un retour est le résultat d'un calcul :

```
In [1]: 1 + 1
Out[1]: 2
```

Un effet modifie l'environnement :

```
In [2]: print('Hello world !')
Hello world !
```

- l'instruction `1 + 1` retourne la valeur 2 et n'a pas d'effet sur l'environnement ;
- l'instruction `print('Hello world !')` retourne la valeur None et a pour effet d'afficher une chaîne de caractères dans le shell.

L'interprète de commande

La fonction print

```
In [3]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

L'interprète de commande

La fonction print

```
In [3]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Le paramètre `file` désigne le «lieu» vers lequel doit être dirigé le flux de caractères à imprimer ; sa valeur par défaut est le shell.

L'interprète de commande

La fonction print

```
In [3]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Le paramètre `file` désigne le «lieu» vers lequel doit être dirigé le flux de caractères à imprimer ; sa valeur par défaut est le shell.
On peut modifier la direction de ce flux :

```
In [4]: print('Hello world !', file=open('essai.txt', 'w'))
```

Cette instruction crée un fichier texte nommé `essai.txt` qui contient la chaîne de caractères `'Hello world !'`.

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
```

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
Out[1]: 3

In [2]: print(1 + 2)
```

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
Out[1]: 3

In [2]: print(1 + 2)
3

In [3]: print(print(1 + 2))
```

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
Out[1]: 3

In [2]: print(1 + 2)
3

In [3]: print(print(1 + 2))
3
None

In [4]: print(1) + 2
```

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
Out[1]: 3

In [2]: print(1 + 2)
3

In [3]: print(print(1 + 2))
3
None

In [4]: print(1) + 2
1

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'

In [5]: print(1) + print(2)
```

L'interprète de commande

Exercice

Compléter l'état du shell après chacune des solicitations :

```
In [1]: 1 + 2
Out[1]: 3

In [2]: print(1 + 2)
3

In [3]: print(print(1 + 2))
3
None

In [4]: print(1) + 2
1

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'

In [5]: print(1) + print(2)
1
2

TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

L'éditeur de texte

Il permet de rédiger des scripts puis de les exécuter, les sauvegarder, etc.

L'éditeur de texte

Il permet de rédiger des scripts puis de les exécuter, les sauvegarder, etc.

```
print('Un nouveau calcul')
print('5 * 3 =', 5*3)
2 + 4
```


```
In [6]: (executing lines 1 to 3 of "<tmp 1>")
Un nouveau calcul
5 * 3 = 15

In [7]:
```

L'exécution d'un script retourne toujours la valeur None.

L'éditeur de texte

Il permet de rédiger des scripts puis de les exécuter, les sauvegarder, etc.
Toute ligne débutant par # est ignorée (c'est un commentaire) :

```
# calcul du nombre de secondes dans une journée
print('une journée a une durée égale à', 60 * 60 * 24, 'secondes')
```

exécution du script

```
In [7]: (executing lines 1 to 2 of "<tmp 1>")
une journée a une durée égale à 86400 secondes

In [8]:
```

La fonction **print** peut posséder plusieurs arguments.

L'éditeur de texte

Exercice

Relire l'aide dédiée à la fonction **print** pour comprendre le rôle des paramètres optionnels sep et end, puis deviner le résultat de l'exécution du script suivant :

```
print(1, 2, 3, sep='+', end='=')
print(6, 5, 4, sep='\n', end='*')
```

L'éditeur de texte

Exercice

Relire l'aide dédiée à la fonction **print** pour comprendre le rôle des paramètres optionnels sep et end, puis deviner le résultat de l'exécution du script suivant :

```
print(1, 2, 3, sep='+', end='=')
print(6, 5, 4, sep='\n', end='*')
```


```
In [8]: (executing lines 1 to 2 of "<tmp 1>")
1+2+3=6
5
4*
In [9]
```

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

La fonction **type** permet de connaître le type d'un objet.

```
In [1]: type(5)
Out[1]: int
```

Le type *int* est utilisé pour représenter les nombres entiers.

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

La fonction **type** permet de connaître le type d'un objet.

```
In [1]: type(5)
Out[1]: int

In [2]: type('LLG')
Out[2]: str
```

Le type *str* permet de représenter des chaînes de caractères.

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

La fonction **type** permet de connaître le type d'un objet.

```
In [1]: type(5)
Out[1]: int

In [2]: type('LLG')
Out[2]: str

In [3]: type(None)
Out[3]: NoneType
```

Le type *NoneType* représente un unique objet : le néant.

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

La fonction `type` permet de connaître le type d'un objet.

```
In [1]: type(5)
Out[1]: int

In [2]: type('LLG')
Out[2]: str

In [3]: type(None)
Out[3]: NoneType

In [4]: type(print)
Out[4]: builtin_function_or_method
```

Tout objet possède un type, y compris les fonctions.

Type d'un objet Python

À tout objet Python est associé :

- un emplacement en mémoire dans lequel est rangé sa représentation binaire ;
- un type qui désigne sa *nature*.

La fonction `id` retourne l'adresse mémoire où se trouve stockée la représentation machine de l'objet.

```
In [5]: id(5)
Out[5]: 4297331360

In [6]: id(None)
Out[6]: 4297071472

In [7]: id(print)
Out[7]: 4298509576
```

La représentation des nombres en Python

Trois types de base peuvent représenter des nombres :

- le type *int* représente des nombres entiers ;
- le type *float* représente des nombres décimaux ;
- le type *complex* représente des nombres complexes.

Ces représentations sont imparfaites : seul une quantité finie de nombres peut être représentée en machine (nous y reviendrons).

La représentation des nombres en Python

Trois types de base peuvent représenter des nombres :

- le type *int* représente des nombres entiers ;
- le type *float* représente des nombres décimaux ;
- le type *complex* représente des nombres complexes.

Ces représentations sont imparfaites : seul une quantité finie de nombres peut être représentée en machine (nous y reviendrons).

En mathématique, $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$; ce n'est pas le cas en informatique. Par exemple, le nombre 2 peut être représenté :

- dans le type *int* sous la forme `2`
- dans le type *float* sous la forme `2.0`
- dans le type *complex* sous la forme `2 + 0j`

La représentation des nombres en Python

Trois types de base peuvent représenter des nombres :

- le type *int* représente des nombres entiers ;
- le type *float* représente des nombres décimaux ;
- le type *complex* représente des nombres complexes.

Ces représentations sont imparfaites : seul une quantité finie de nombres peut être représentée en machine (nous y reviendrons).

En mathématique, $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$; ce n'est pas le cas en informatique. Par exemple, le nombre 2 peut être représenté :

- dans le type *int* sous la forme `2`
- dans le type *float* sous la forme `2.0`
- dans le type *complex* sous la forme `2 + 0j`

Les algorithmes utilisés pour réaliser une opération arithmétique diffèrent suivant le type des objets :

```
In [1]: 1 + 2 - 3
Out[1]: 0

In [2]: 0.1 + 0.2 - 0.3
Out[2]: 5.551115123125783e-17
```

La représentation des nombres en Python

Trois types de base peuvent représenter des nombres :

- le type *int* représente des nombres entiers ;
- le type *float* représente des nombres décimaux ;
- le type *complex* représente des nombres complexes.

Ces représentations sont imparfaites : seul une quantité finie de nombres peut être représentée en machine (nous y reviendrons).

En mathématique, $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$; ce n'est pas le cas en informatique. Par exemple, le nombre 2 peut être représenté :

- dans le type *int* sous la forme `2`
- dans le type *float* sous la forme `2.0`
- dans le type *complex* sous la forme `2 + 0j`

Dans certains langages il est impossible d'effectuer une opération mêlant des objets de types différents. En Python, si cela s'avère nécessaire la conversion de type est automatique dans le sens :

$$\texttt{int} \longrightarrow \texttt{float} \longrightarrow \texttt{complex}$$

mais jamais dans le sens contraire.

La représentation des nombres en Python

Exemples

```
In [1]: 4 * 5
Out[1]: 20
```

Les deux arguments sont de type *int*, le résultat aussi.

La représentation des nombres en Python

Exemples

```
In [1]: 4 * 5
Out[1]: 20

In [2]: 23 / 3
Out[2]: 7.666666666666667
```

Les deux arguments sont de type *int*, le résultat de type *float*.

La représentation des nombres en Python

Exemples

```
In [1]: 4 * 5
Out[1]: 20

In [2]: 23 / 3
Out[2]: 7.666666666666667

In [3]: 4 * 5.
Out[3]: 20.0
```

Le premier argument est de type *int*, le second de type **float** ; le résultat est de type *float*.

La représentation des nombres en Python

Exemples

```
In [1]: 4 * 5
Out[1]: 20

In [2]: 23 / 3
Out[2]: 7.666666666666667

In [3]: 4 * 5.
Out[3]: 20.0

In [4]: 24 / 4
Out[4]: 6.0
```

Les deux arguments sont de type *int*, le résultat de type *float*.

La représentation des nombres en Python

Exemples

Le quotient et le reste d'une division euclidienne se notent respectivement // et % :

```
In [5]: 23 // 3
Out[5]: 7

In [6]: 23 % 3
Out[6]: 2

In [7]: 24 // 4
Out[7]: 6

In [8]: 24 % 4
Out[8]: 0
```

Ces opérations sont à privilégier lorsqu'on utilise des données de type *int*.

La représentation des nombres en Python

Conversion explicite de type

Les inclusions mathématiques $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$ permettent la conversion automatique des types :

$$\textit{int} \longrightarrow \textit{float} \longrightarrow \textit{complex}$$

La représentation des nombres en Python

Conversion explicite de type

Les inclusions mathématiques $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$ permettent la conversion automatique des types :

$$int \longrightarrow float \longrightarrow complex$$

les fonctions `int`, `float`, `complex` réalisent *dans certains cas* une conversion explicite :

```
In [9]: int(2.3333)
Out[9]: 2

In [10]: int(-2.3333)
Out[10]: -2
```

Attention, `int(x)` ne retourne pas la partie entière de x si ce dernier est négatif.

La représentation des nombres en Python

Conversion explicite de type

Les inclusions mathématiques $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$ permettent la conversion automatique des types :

$$\textit{int} \longrightarrow \textit{float} \longrightarrow \textit{complex}$$

les fonctions **int**, **float**, **complex** réalisent *dans certains cas* une conversion explicite :

```
In [11]: float(25)
Out[11]: 25.0

In [12]: complex(2)
Out[12]: (2+0j)
```

(exemples de conversions naturelles).

La représentation des nombres en Python

Conversion explicite de type

Les inclusions mathématiques $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$ permettent la conversion automatique des types :

$$\textit{int} \longrightarrow \textit{float} \longrightarrow \textit{complex}$$

les fonctions **int**, **float**, **complex** réalisent *dans certains cas* une conversion explicite :

```
In [13]: float(9999999999999999)
Out[13]: 1e+16

In [14]: int(1e+16)
Out[14]: 10000000000000000
```

Attention, ces conversions peuvent être irréversibles.

La représentation des nombres en PYTHON

Conversion explicite de type

Les inclusions mathématiques $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$ permettent la conversion automatique des types :

$$\textit{int} \longrightarrow \textit{float} \longrightarrow \textit{complex}$$

les fonctions **int**, **float**, **complex** réalisent *dans certains cas* une conversion explicite :

```
In [15]: float(2 + 0j)
TypeError: can't convert complex to float

In [16]: (2 + 0j).real
Out[16]: 2.0

In [17]: (2+0j).imag
Out[17]: 0.0
```

Il n'est pas possible de convertir un type *complex* en type *float* ou *int*, mais on peut prendre la partie réelle pour obtenir un objet de type *float*.

Calculer avec Python

Fonctions mathématiques supplémentaires

Les modules math et numpy fournissent entres autres :

- les fonctions trigonométriques sin, cos, tan ;
- les fonctions exponentielle exp et logarithme log ;
- la fonction racine carrée sqrt.

Calculer avec Python

Fonctions mathématiques supplémentaires

Les modules math et numpy fournissent entres autres :

- les fonctions trigonométriques sin, cos, tan ;
- les fonctions exponentielle exp et logarithme log ;
- la fonction racine carrée sqrt.

```
In [15]: import numpy as np

In [16]: np.sin(1.571)
Out[16]: 0.9999997925861284

In [17]: np.sqrt(2)
Out[17]: 1.4142135623730951

In [18]: np.pi
Out[18]: 3.141592653589793
```

Calculer avec Python

Fonctions mathématiques supplémentaires

Les modules math et numpy fournissent entres autres :

- les fonctions trigonométriques sin, cos, tan ;
- les fonctions exponentielle exp et logarithme log ;
- la fonction racine carrée sqrt.

```
In [15]: import numpy as np

.........................

In [19]: from math import exp

In [20]: exp(1)                  # la fonction exp du module math
Out[20]: 2.718281828459045

In [21]: np.exp(1)               # la fonction exp du module numpy
Out[21]: 2.7182818284590451
```

Booléens

Le type *bool* ne comporte que deux objets : `True` et `False`.
À ce type sont associés trois opérateurs : **`not`**, **`and`** et **`or`** définis par :

not	
False	True
True	False

and	False	True
False	False	False
True	False	True

or	False	True
False	False	True
True	True	True

Booléens

Le type *bool* ne comporte que deux objets : `True` et `False`.
À ce type sont associés trois opérateurs : **`not`**, **`and`** et **`or`** définis par :

not	
False	True
True	False

and	False	True
False	False	False
True	False	True

or	False	True
False	False	True
True	True	True

Par ailleurs, un certain nombre d'opérateurs sont définis sur d'autres types (en particulier les types de nombres) et à valeurs dans le type *bool*.

```
In [1]: (4 + 3) < 11 and not 'alpha' > 'omega'
Out[1]: True

In [2]: (1 + 1 == 3) != ('Henri 4' > 'Louis-le-Grand')
Out[2]: False
```

On retiendra qu'*évaluer* une expression booléenne n'est que le résultat d'un calcul.

Variables

Les données calculées peuvent être mémorisées à l'aide de *variables* : on attribue un *nom* à cette variable et on lui affecte une valeur.

```
In [1]: largeur = 12.45

In [2]: longueur = 42.18

In [3]: aire = longueur * largeur

In [4]: print("l'aire du rectangle est égale à", aire)
l'aire du rectangle est égale à 525.141
```

Variables

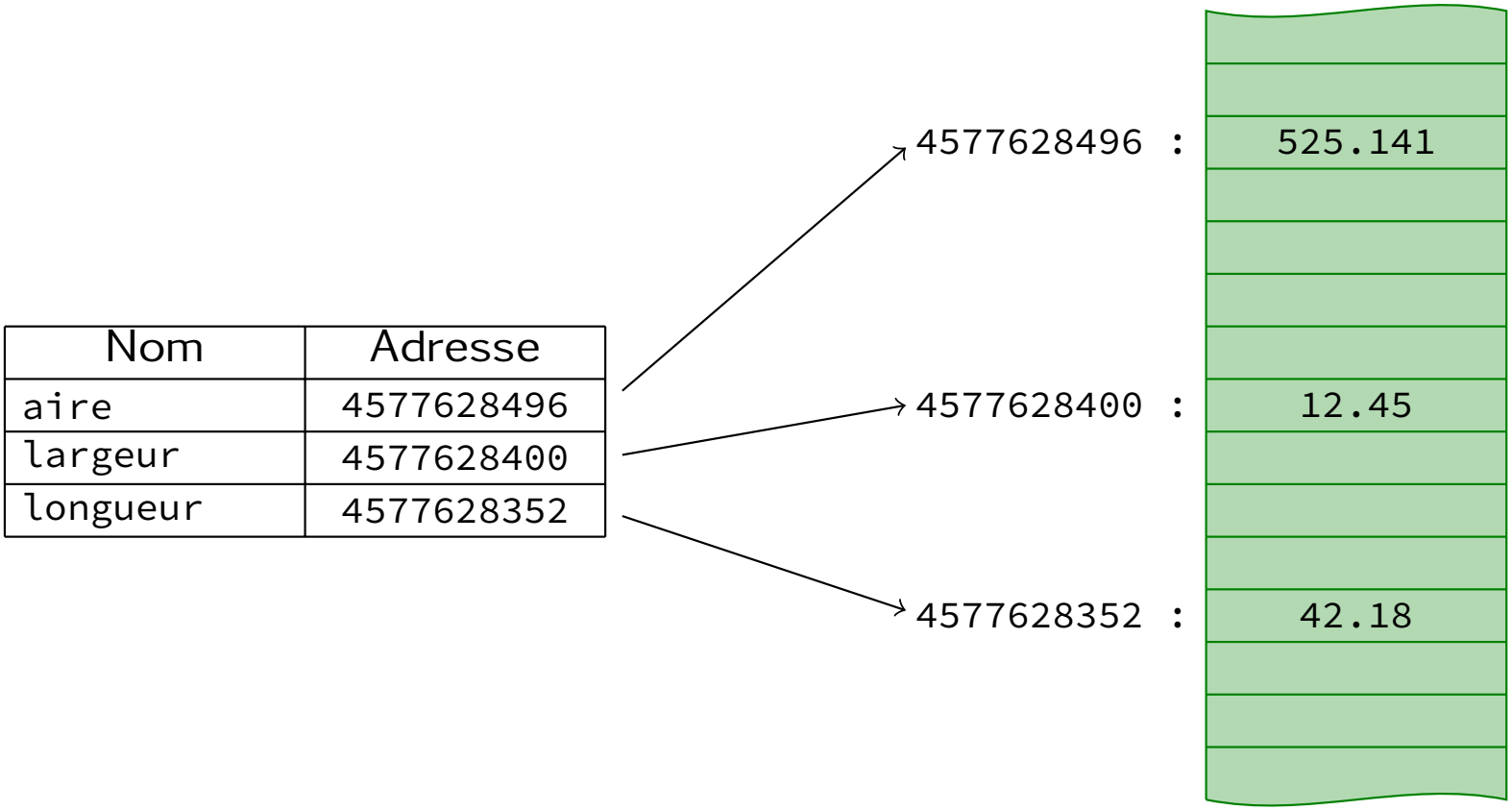
Les données calculées peuvent être mémorisées à l'aide de *variables* : on attribue un *nom* à cette variable et on lui affecte une valeur.

```
In [1]: largeur = 12.45

In [2]: longueur = 42.18

In [3]: aire = longueur * largeur

In [4]: print("l'aire du rectangle est égale à", aire)
l'aire du rectangle est égale à 525.141
```

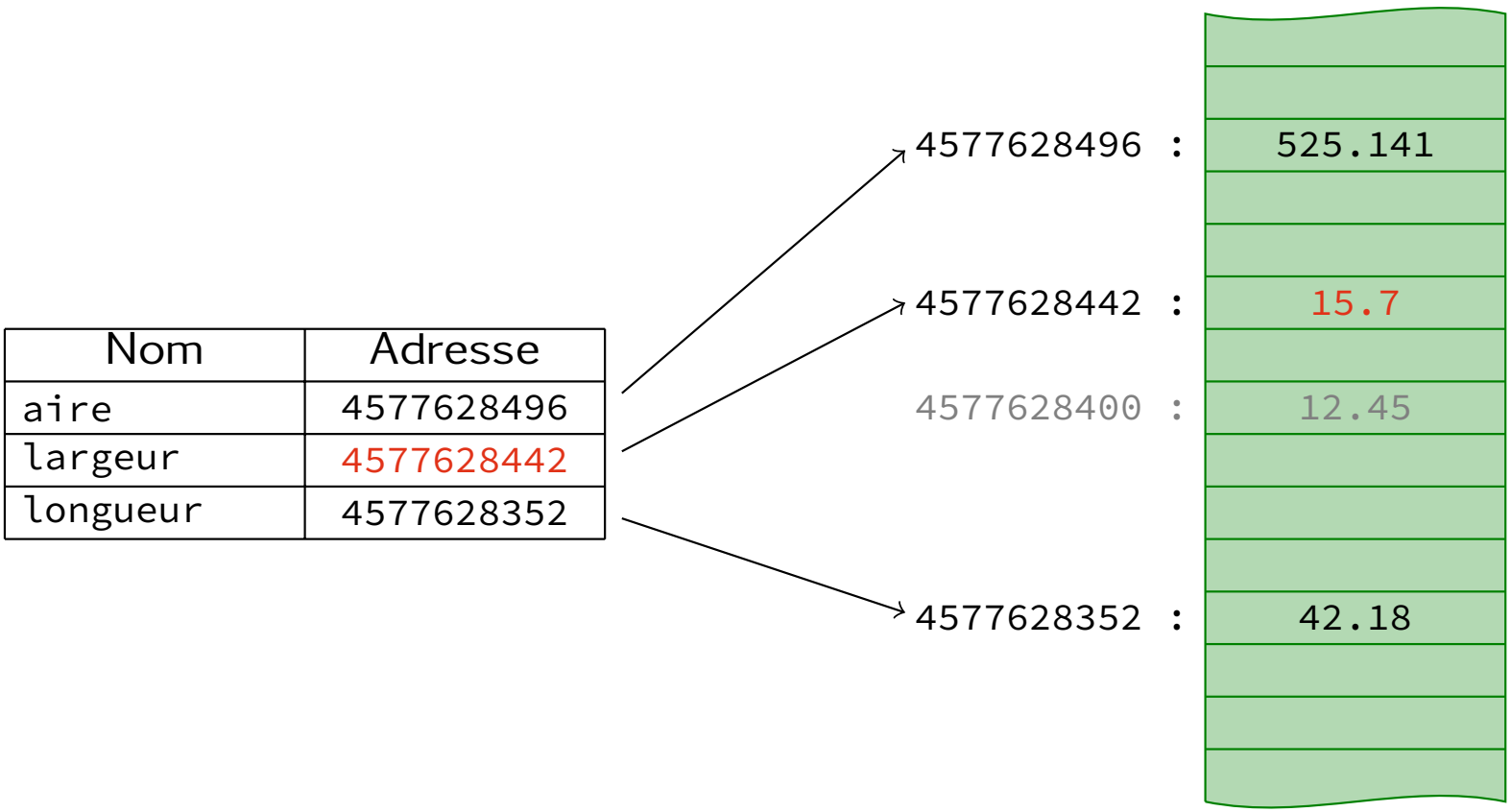
Nom	Adresse	Valeur
aire	4577628496	525.141
largeur	4577628400	12.45
longueur	4577628352	42.18

```
In [5]: id(largeur)
Out[5]: 4577628400
```

Variables


Nom
Adresse
aire
4577628496
largeur
4577628400
longueur
4577628352
4577628496 :
525.141
4577628400 :
12.45
4577628352 :
42.18

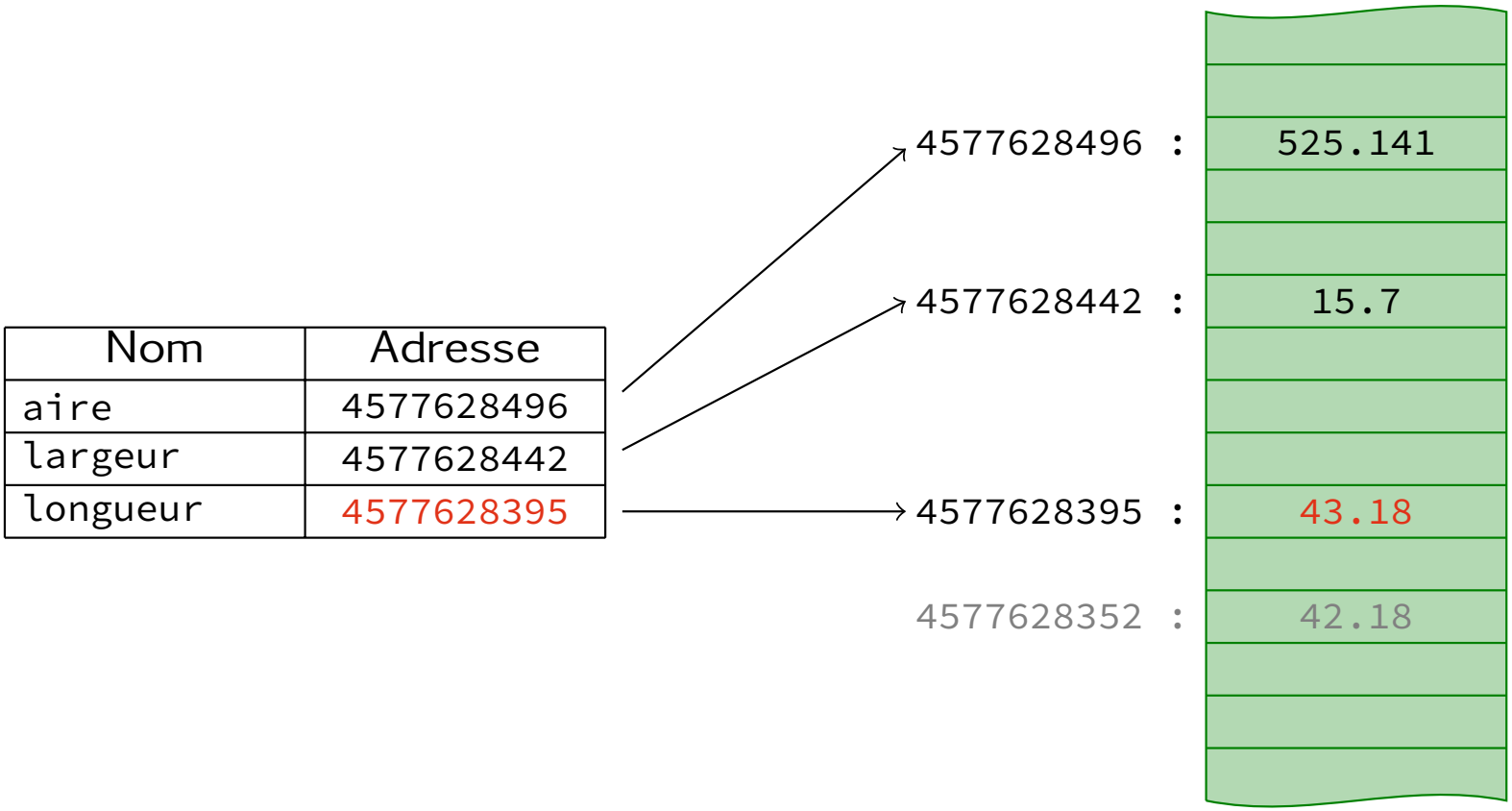
Variables



Une fois définie, la valeur de la variable peut être modifiée, toujours à l'aide de l'opérateur d'affectation.

```
In [6]: largeur = 15.7
```

Variables



Le contenu d'une variable peut servir à sa propre modification :

```
In [7]: longueur = longueur + 1
```

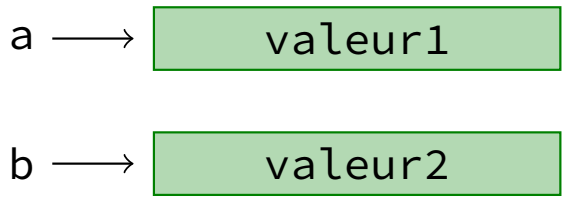
ou

```
In [7]: longueur += 1
```

Variables

Affectations parallèles

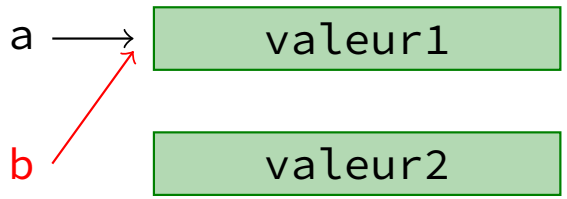
Un problème classique : comment permuter le contenu de deux variables ?



Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



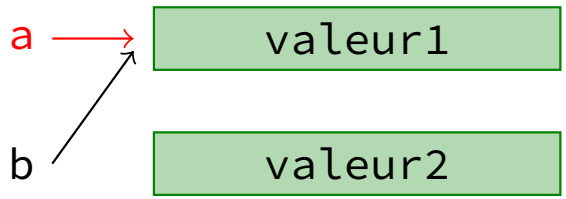
Première tentative :

```
In [1]: b = a
```

Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



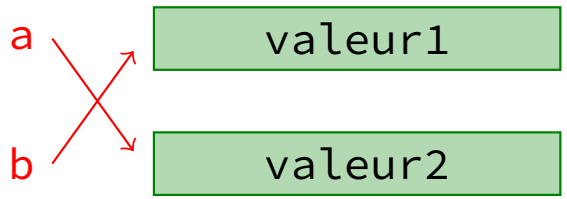
Première tentative : échec

```
In [1]: b = a

In [2]: a = b
```

Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



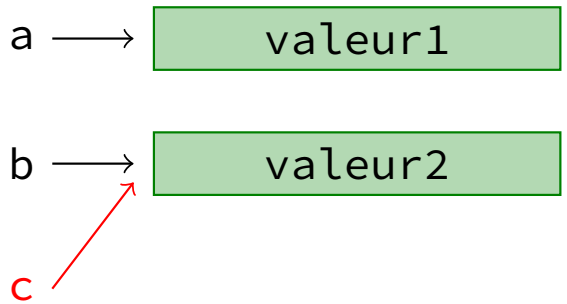
Avec une affectation parallèle :

```
In [1]: a, b = b, a
```

Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



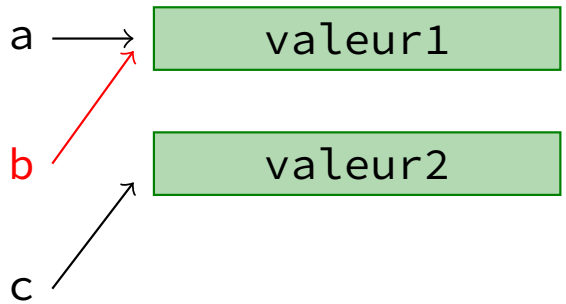
Sans affectation parallèle, il faut utiliser une variable auxiliaire :

```
In [1]: c = b
```

Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



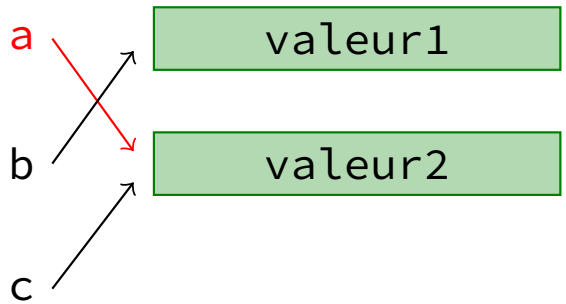
Sans affectation parallèle, il faut utiliser une variable auxiliaire :

```
In [1]: c = b

In [2]: b = a
```

Variables

Affectations parallèles

Un problème classique : comment permuter le contenu de deux variables ?



Sans affectation parallèle, il faut utiliser une variable auxiliaire :

```
In [1]: c = b

In [2]: b = a

In [3]: a = c
```

Égalité de valeur, égalité physique

De ce mécanisme résulte l'existence de deux sortes d'égalités entre variables :

- égalité de valeur lorsque deux variables *a* et *b* référencent la même valeur à deux emplacements mémoires pouvant être différents ;
- égalité physique lorsque deux variables *a* et *b* référencent le même emplacement mémoire (ce qui implique l'égalité de valeur).

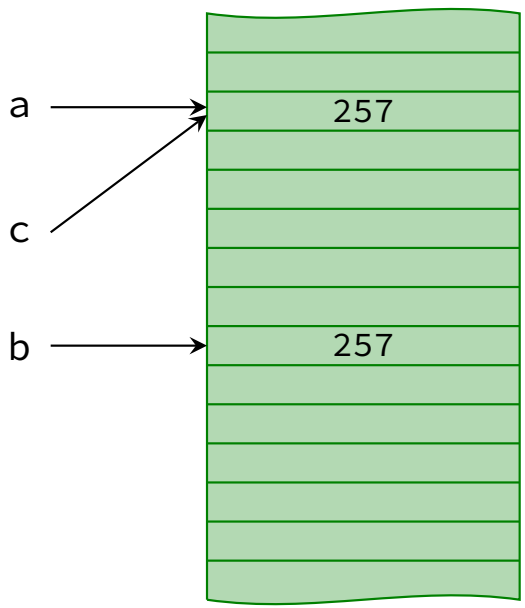
Égalité de valeur, égalité physique

De ce mécanisme résulte l'existence de deux sortes d'égalités entre variables :

- égalité de valeur lorsque deux variables *a* et *b* référencent la même valeur à deux emplacements mémoires pouvant être différents ;
- égalité physique lorsque deux variables *a* et *b* référencent le même emplacement mémoire (ce qui implique l'égalité de valeur).

L'opérateur qui teste l'égalité de valeur est == ; l'opérateur qui teste l'égalité physique se nomme `is`.

```
In [11]: a = 257

In [12]: b = 257

In [13]: c = a

In [13]: a == b
Out[13]: True

In [14]: a is b
Out[13]: False

In [14]: a is c
Out[13]: True
```



Chaînes de caractères

Les données alphanumériques sont appelées des *chaînes de caractères* : c'est le type str ; elles sont délimitées par des " ou par des '.

```
In [1]: "aujourd'hui"
Out[1]: "aujourd'hui"

In [2]: 'et demain'
Out[2]: 'et demain'
```

Chaînes de caractères

Les données alphanumériques sont appelées des *chaînes de caractères* : c'est le type str ; elles sont délimitées par des " ou par des '.

```
In [1]: "aujourd'hui"
Out[1]: "aujourd'hui"

In [2]: 'et demain'
Out[2]: 'et demain'
```

Le caractère spécial de passage à la ligne est représenté par \n :

```
In [3]: print("un passage\n à la ligne")
un passage
 à la ligne
```

Chaînes de caractères

Opération sur les chaînes de caractères

L'opérateur + réalise l'opération de concaténation, * réalise la duplication :

```
In [4]: chn = "Hello "

In [5]: chn += 'world !'     # équivalent à chn = chn + 'world !'

In [6]: print(chn)
Hello world !

In [7]: chn * 3              # équivalent à chn + chn + chn
Out[7]: 'Hello world !Hello world !Hello world !'
```

Chaînes de caractères

Opération sur les chaînes de caractères

Attention à ne pas confondre chaîne de caractère et nombre.

```
In [7]: '123' + '1'          # concaténation de deux chaînes
Out[7]: '1231'

In [8]: 123 + 1              # addition de deux entiers
Out[8]: 124

In [9]: '123' + 1
TypeError: cannot concatenate 'str' and 'int' objects
```

Chaînes de caractères

Opération sur les chaînes de caractères

Attention à ne pas confondre chaîne de caractère et nombre.

```
In [7]: '123' + '1'          # concaténation de deux chaînes
Out[7]: '1231'

In [8]: 123 + 1              # addition de deux entiers
Out[8]: 124

In [9]: '123' + 1
TypeError: cannot concatenate 'str' and 'int' objects
```

Les fonctions `int` et `str` permettent de forcer la conversion de type :

```
In [10]: int('123') + 1
Out[10]: 124

In [11]: '123' + str(1)
Out[11]: '1231'
```

Chaînes de caractères

Opération sur les chaînes de caractères

Attention à ne pas confondre chaîne de caractère et nombre.

```
In [7]: '123' + '1'          # concaténation de deux chaînes
Out[7]: '1231'

In [8]: 123 + 1              # addition de deux entiers
Out[8]: 124

In [9]: '123' + 1
TypeError: cannot concatenate 'str' and 'int' objects
```

Les fonctions **int** et **str** permettent de forcer la conversion de type :

```
In [10]: int('123') + 1
Out[10]: 124

In [11]: '123' + str(1)
Out[11]: '1231'
```

De même, la fonction **float** convertit une donnée adéquate (entier ou chaîne de caractères) en un nombre flottant.

Chaînes de caractères

Accès aux caractères individuels

Chaque caractère est accessible par son rang.

```
In [12]: ch = 'Louis-Le-Grand'

In [13]: ch[4]
Out[13]: 's'

In [14]: ch[0] + ch[6] + ch[9]
Out[14]: 'LLG'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	o	u	i	s	-	L	e	-	G	r	a	n	d

Chaînes de caractères

Accès aux caractères individuels

Chaque caractère est accessible par son rang.

```
In [12]: ch = 'Louis-Le-Grand'

In [13]: ch[4]
Out[13]: 's'

In [14]: ch[0] + ch[6] + ch[9]
Out[14]: 'LLG'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	o	u	i	s	-	L	e	-	G	r	a	n	d
−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

Il est aussi possible d'indexer les caractères par des entiers négatifs :

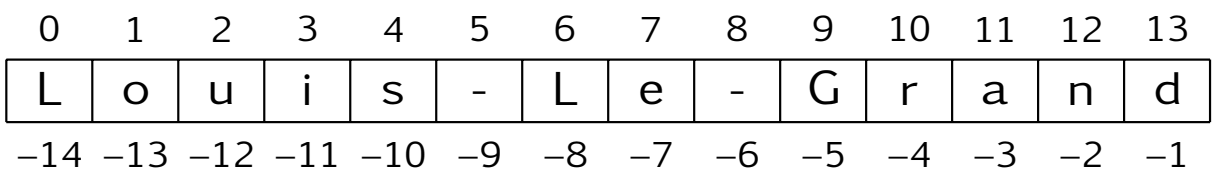
```
In [15]: print(ch[-8] * 2 + ch[-5])
LLG
```

Chaînes de caractères

Accès aux caractères individuels

Chaque caractère est accessible par son rang.

```
In [12]: ch = 'Louis-Le-Grand'

In [13]: ch[4]
Out[13]: 's'

In [14]: ch[0] + ch[6] + ch[9]
Out[14]: 'LLG'
```



La fonction `len` permet de calculer la longueur d'une chaîne de caractères :

```
In [16]: len(ch)
Out[16]: 14
```

Chaînes de caractères

Slicing

0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	o	u	i	s	-	L	e	-	G	r	a	n	d
−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

La syntaxe `ch[i:j]` permet d'extraire une portion de chaîne :

```
In [17]: ch[3:-3]
Out[17]: 'is-Le-Gr'
```

Chaînes de caractères

Slicing

0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	o	u	i	s	-	L	e	-	G	r	a	n	d
−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

La syntaxe `ch[i:j]` permet d'extraire une portion de chaîne :

```
In [17]: ch[3:-3]
Out[17]: 'is-Le-Gr'
```

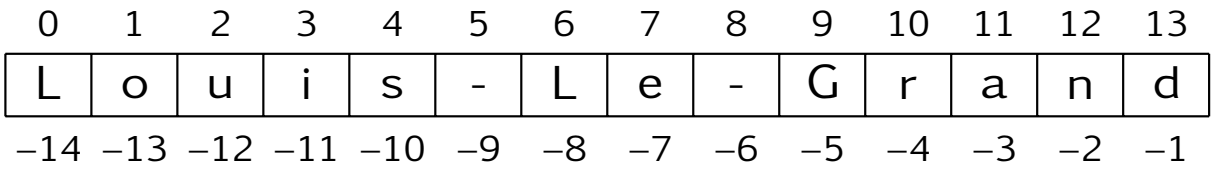
Par défaut `i` vaut 0 et `j` la longueur de la chaîne :

```
In [18]: ch[6:] + ch[:6]
Out[18]: 'Le-GrandLouis-'
```

- `ch[:k]` renvoie les k premiers caractères de `ch` ;
- `ch[-k:]`renvoie les k derniers caractères de `ch`.

Chaînes de caractères

Slicing



Un troisième paramètre donne le pas de la sélection :

```
In [19]: ch[::2]
Out[19]: 'LusL-rn'

In [20]: ch[1::2]
Out[20]: 'oi-eGad'
```

Chaînes de caractères

Slicing

0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	o	u	i	s	-	L	e	-	G	r	a	n	d
−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

Un troisième paramètre donne le pas de la sélection :

```
In [19]: ch[::2]
Out[19]: 'LusL-rn'

In [20]: ch[1::2]
Out[20]: 'oi-eGad'
```

Enfin, le pas peut aussi prendre des valeurs négatives :

```
In [21]: ch[::-1]
Out[21]: 'dnarG-eL-siuoL'
```

On obtient ainsi aisément l'image miroir d'une chaîne de caractères.

Exercice

Mélange de Monge

Le *mélange de* Monge d'un paquet de cartes numérotées de 1 à $2n$ consiste à démarrer un nouveau paquet avec la carte 1, à placer la carte 2 au dessus de ce nouveau paquet, puis la carte 3 au dessous du nouveau paquet et ainsi de suite en plaçant les cartes paires au dessus du nouveau paquet et les cartes impaires au dessous.

$$(1, 2, 3, \ldots, 2n) \longrightarrow (2n, 2n-2, \ldots, 4, 2, 1, 3, 5, \ldots, 2n-3, 2n-1)$$

Réaliser en une ligne Python le calcul d'un mélange de Monge d'une chaîne de caractères s.

Exercice

Mélange de Monge

Le *mélange de* Monge d'un paquet de cartes numérotées de 1 à $2n$ consiste à démarrer un nouveau paquet avec la carte 1, à placer la carte 2 au dessus de ce nouveau paquet, puis la carte 3 au dessous du nouveau paquet et ainsi de suite en plaçant les cartes paires au dessus du nouveau paquet et les cartes impaires au dessous.

$$(1, 2, 3, \ldots, 2n) \longrightarrow (2n, 2n-2, \ldots, 4, 2, 1, 3, 5, \ldots, 2n-3, 2n-1)$$

Réaliser en une ligne Python le calcul d'un mélange de Monge d'une chaîne de caractères s.

```
In [1]: s = '12345678'

In [2]: s[-1::-2] + s[0::2]
Out[2]: '86421357'
```