Cours d'algorithmique et de programmation orientée objet

UED SIM, module «*Systèmes Informatiques*»

Jean-luc CHARLES (jean-luc.charles@ensam.eu)
Éric DUCASSE (eric.ducasse@ensam.eu)

Arts & Métiers ParisTech *Bordeaux*

Un petit exemple introductif

Algorithme: Confection d'un moelleux au chocolat

Entrée(s): chocolat, oeufs, beurre, farine, sucre, four, casserole, saladier, moule
Retour(s): gâteau

```
1  four.allumer(200 °C)
2  chocolat.mesurer(200 g)
3  casserole.ajouter(chocolat)
4  beurre.mesurer(170 g)
5  casserole.ajouter(beurre)
6  casserole.chauffer(600 °C)
7  Tant que : casserole.contenu().estfondu()==False
8      |casserole.mélanger()
9  casserole.éteindre()
10 oeufs.mesurer(4)
11 saladier.ajouter(oeufs)
12 saladier.fouetter()
13 Pour nocuillere parcourant la liste ⟦1, 10⟧ faire :
14     |sucre.mesurer(15 g)
15     |saladier.ajouter(sucre)
16     |saladier.fouetter()
```

Un petit exemple introductif

Algorithme: Confection d'un moelleux au chocolat (suite)

```
17 farine.mesurer(50 g)
18 saladier.ajouter(farine)
19 saladier.melanger()
20 saladier.incorporer(casserole.contenu())
21 moule.beurrer()
22 moule.ajouter(saladier.contenu())
23 Tant que : four.temperature() < 200 °C
24       |attendre()
25 four.entrer(moule)
26 attendre(12′)
27 four.éteindre()
28 four.sortir(moule)
29 Retour : moule.contenu()
```

Table des matières

Didacticiels et bibliothèques :

- `http://fr.openclassrooms.com/`
- `http://www.developpez.com/`
- `http://www.framasoft.net/article1971.html` («*Apprendre à programmer avec Python*», de Gérard Swinnen)
- `http://docs.python.org/library/` ("*The Python Standard Library* ")
- `http://qt-project.org/doc/qt-5.1/qtdoc/reference-overview.html` ("*The Qt Reference Documentation* ")
- ...

Erreur en *Python*™ : `NameError: name 'a' is not defined`

	Instructions	Sortie (Console)
1	# en-tête	
2	# -*- coding : latin-1 -*-	
3	# MAIN (Programme principal)	
4	print "Valeur de a : ", a	

Erreur de compilation en C++ : `'a' was not declared in this scope`

	Instructions	Sortie (Console)
1	// en-tête	
2	#include <iostream>	
3	using namespace std ;	
4	// MAIN (Programme principal)	
5	int main()	
6	{	
7	cout << "Valeur de a : " << a << endl ;	
8	return 0 ;	
9	}	

Mathematica® : 'a' est un symbole

	Instructions	Sortie (Console)
1	`Print["Valeur de a : ", a];`	`Valeur de a : a`

Qu'est-ce qu'une variable ou plus généralement un objet ?

- Un **type** et une **adresse** définissant un **emplacement en mémoire**
  Types de base : entiers ('int', 'long'...), flottants ('float', 'double'...), booléens , caractères (ASCII, Unicode...).
- Un **contenu** (données)
- Un ou plusieurs **identifiants** (noms)

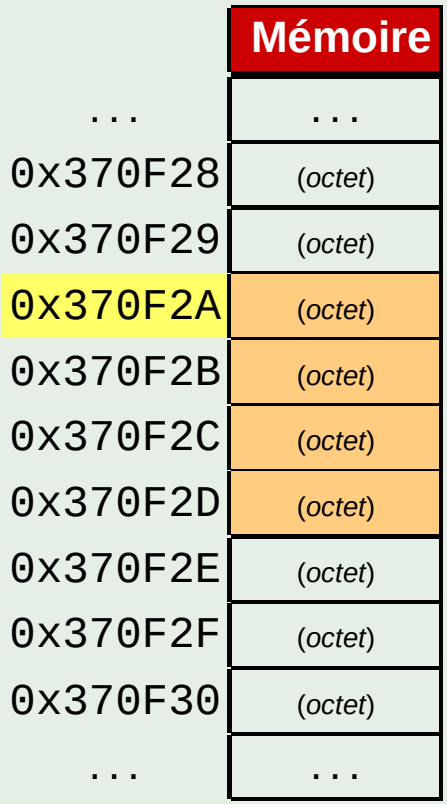
Typage statique

Notion usuelle de « *variable* »

1. On déclare qu'un identifiant donné correspond à un type donné
2. Un emplacement mémoire (adresse, nombre d'octets...) est alloué
3. Le contenu est défini (en général par une affectation) et peut être ensuite modifié

Typage dynamique

1. On définit un objet d'un type et d'un contenu donnés
2. Un emplacement mémoire (adresse, nombre d'octets...) est alloué
3. On attribue un identifiant à cet objet (en général par une affectation)
4. D'autres identifiants peuvent éventuellement attribués à cet objet (*alias*)



Typage statique / dynamique : l'influence sur l'affectation

Algorithme: Affectation et adresse

```
1 x ← 12.3456
2 Print x, adresse de x
3 y ← x
4 Print y, adresse de y
5 x ← 65.4321
6 Print x, adresse de x
```

Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456;`	
2	`cout << "x : val. " << x << ";\n adr. " << &x << endl;`	`x : val. 12.3456;` `adr. 0x28ff04`
3	`double y = x;`	
4	`cout << "y : val. " << y << ";\n adr. " << &y << endl;`	`y : val. 12.3456;` `adr. 0x28fefc`
5	`x = 65.4321;`	
6	`cout << "x : val. " << x << ";\n adr. " << &x << endl;`	`x : val. 65.4321;` `adr. 0x28ff04`

Typage statique / dynamique : l'influence sur l'affectation

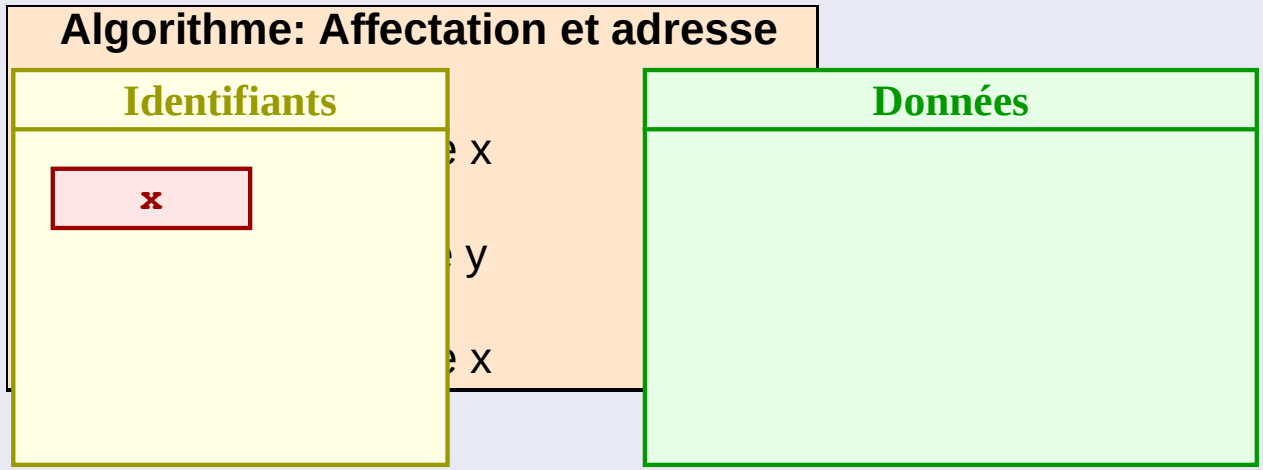
Algorithme: Affectation et adresse

```
1 x ← 12.3456
2 Print x, adresse de x
3 y ← x
4 Print y, adresse de y
5 x ← 65.4321
6 Print x, adresse de x
```

Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

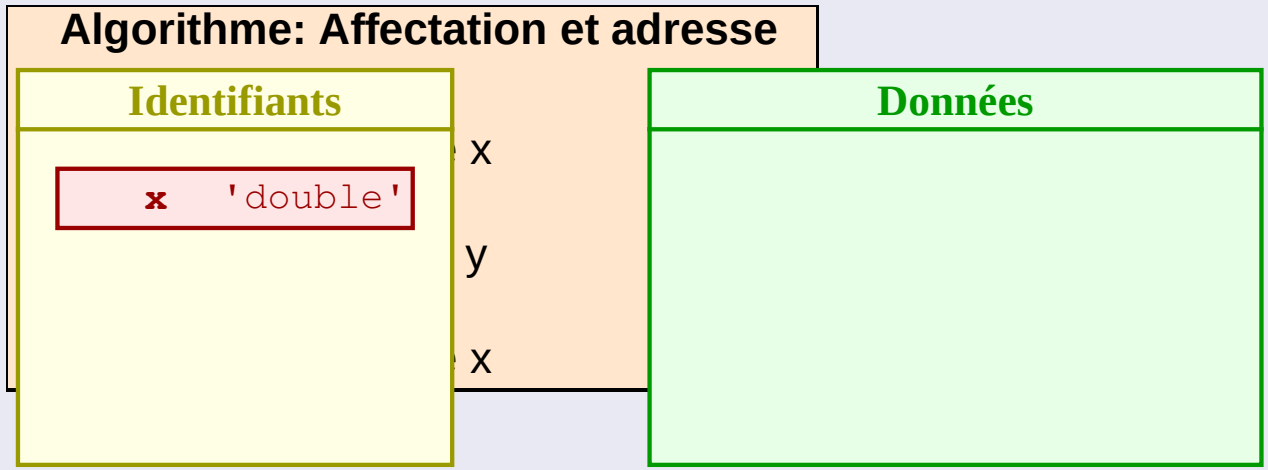
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

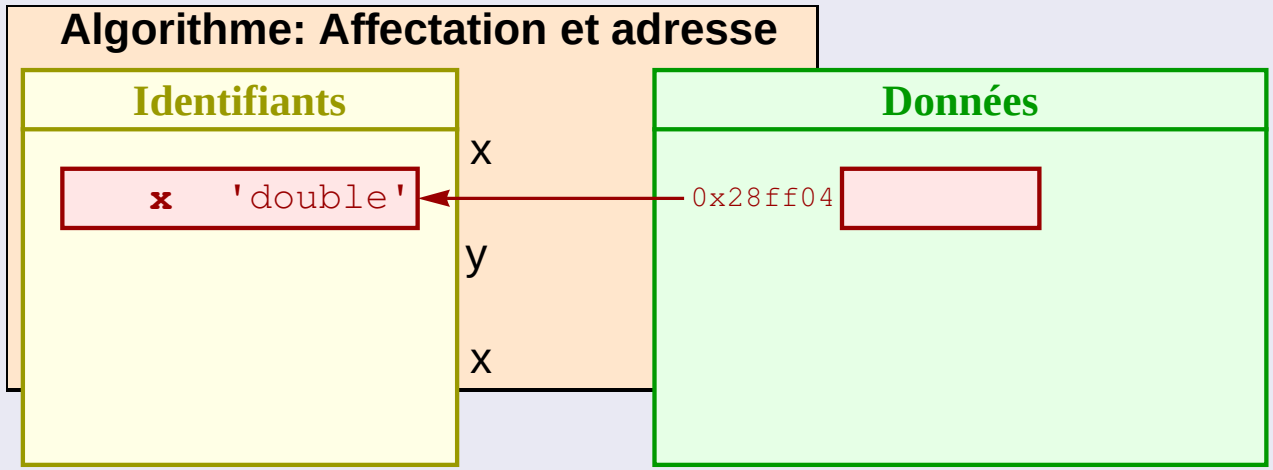
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

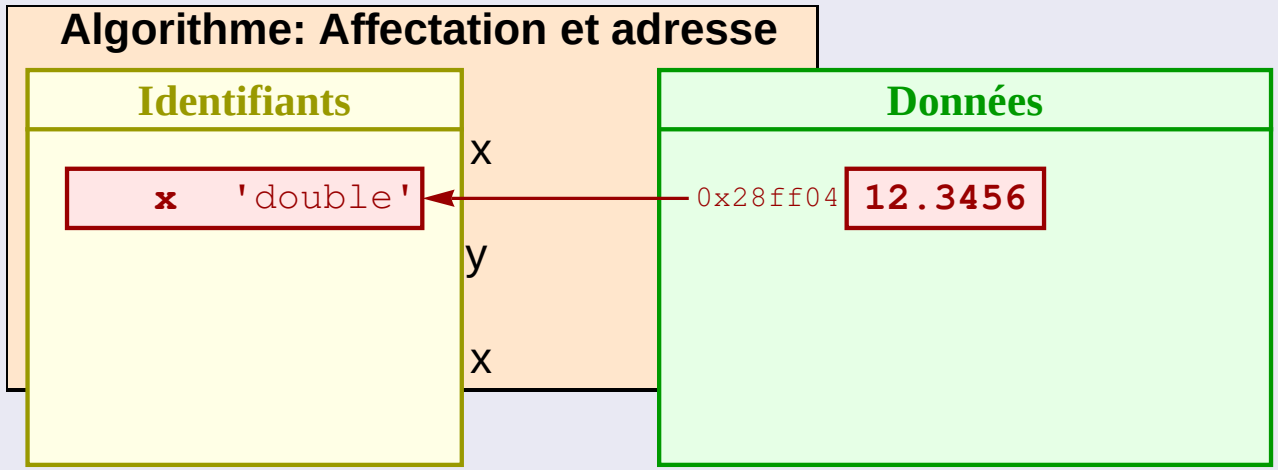
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

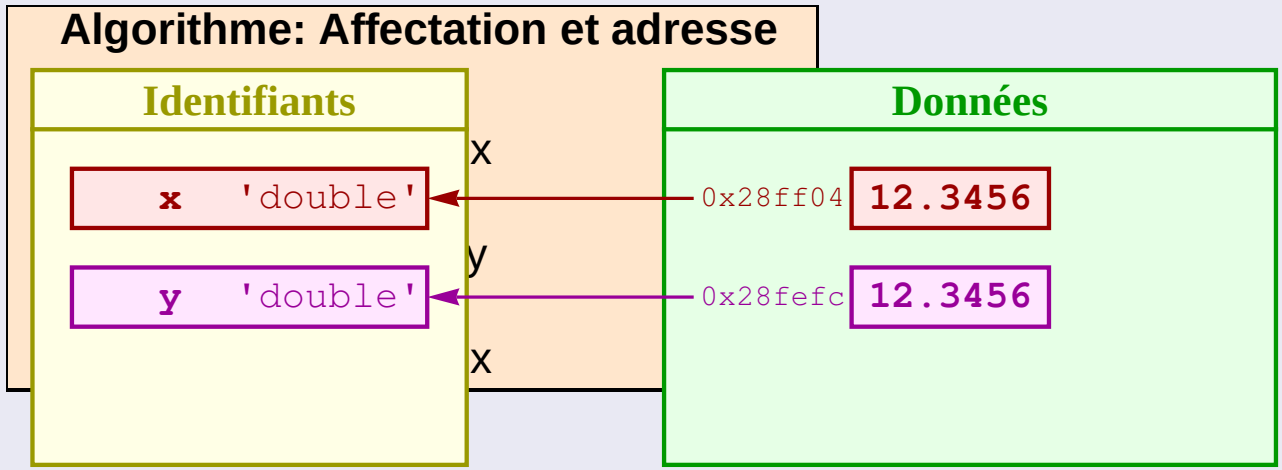
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

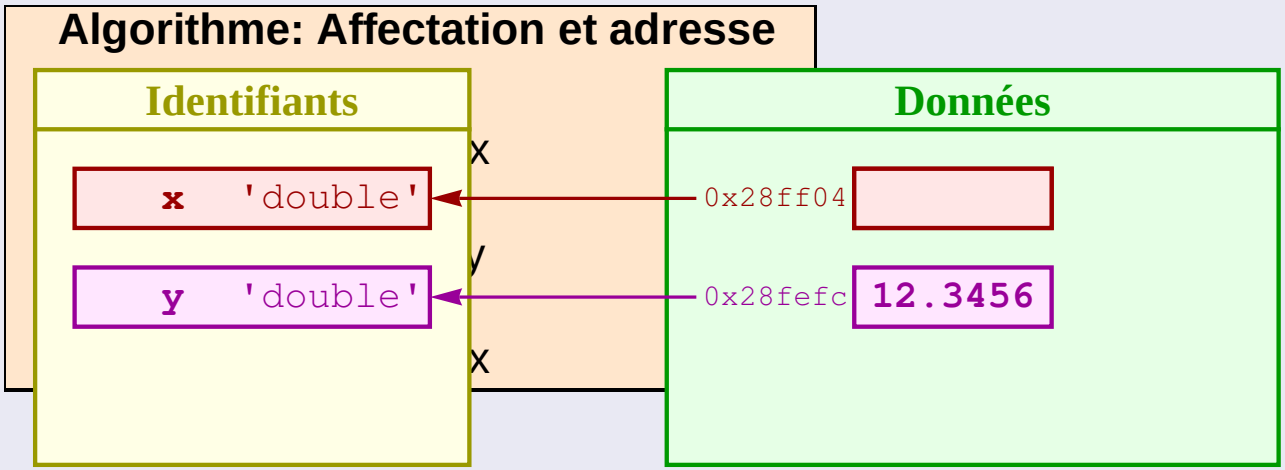
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

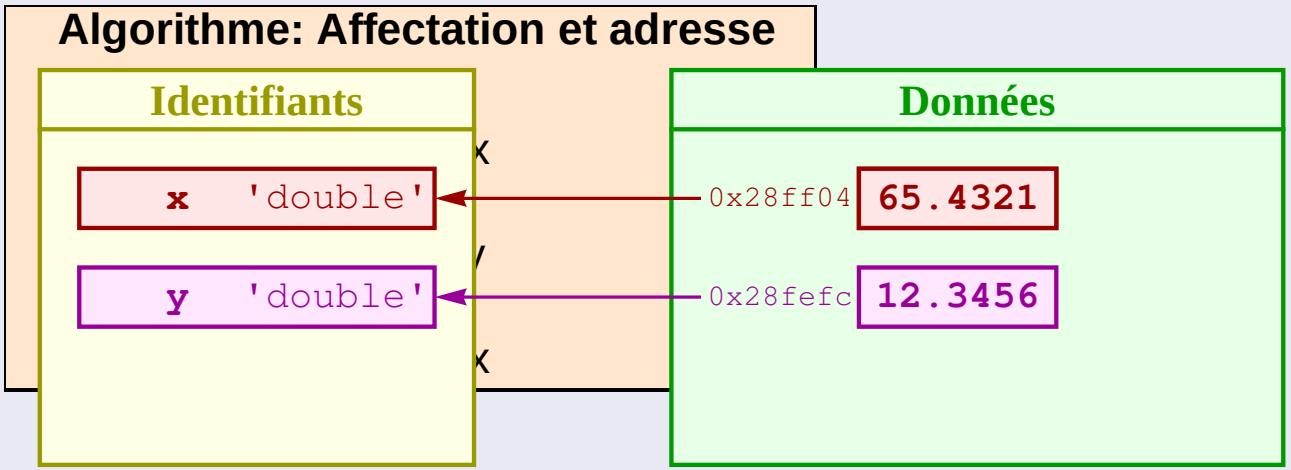
Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

Typage statique / dynamique : l'influence sur l'affectation



Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`double x = 12.3456 ;`	
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 12.3456 ;` `adr. 0x28ff04`
3	`double y = x ; // Affectation par valeur`	
4	`cout << "y : val. " << y << " ;\n adr. " << &y << endl ;`	`y : val. 12.3456 ;` `adr. 0x28fefc`
5	`x = 65.4321 ;`	
6	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	`x : val. 65.4321 ;` `adr. 0x28ff04`

Typage statique / dynamique : l'influence sur l'affectation

Algorithme: Affectation et adresse

```
1 x ← 12.3456
2 Print x, adresse de x
3 y ← x
4 Print y, adresse de y
5 x ← -65
6 Print x, adresse de x
```

Typage dynamique, exemple de *Python*™

Instructions	Sortie (Console)
1 x = 12.3456 2 print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x) 3 y = x 4 print "y : val. ", y, " ;\n adr. ", hex(id(y)) 5 x = -65 6 print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x) 7 print "y : val. ", y, " ;\n adr. ", hex(id(y))	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'> y : val. 12.3456 ; adr. 0x3bca78 x : val. -65 ; adr. 0x3ac708 <type 'int'> y : val. 12.3456 ; adr. 0x3bca78

Typage statique / dynamique : l'influence sur l'affectation

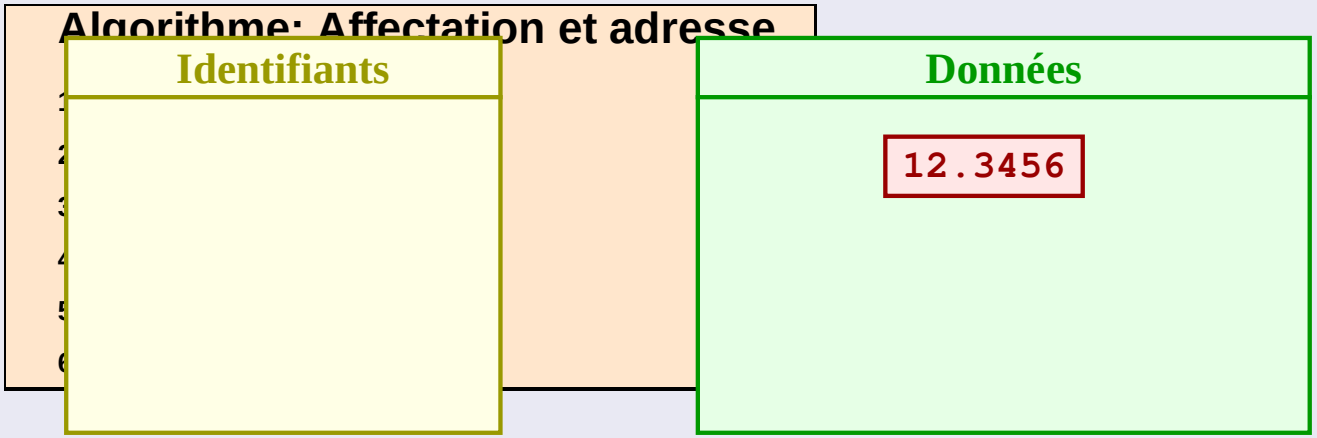
Algorithme: Affectation et adresse

```
1 x ← 12.3456
2 Print x, adresse de x
3 y ← x
4 Print y, adresse de y
5 x ← -65
6 Print x, adresse de x
```

Typage dynamique, exemple de *Python*™

Instructions	Sortie (Console)
1 x = 12.3456	x : val. 12.3456; adr. 0x3bca78 <type 'float'>
2 print "x : val. ", x, ";\n adr. ", hex(id(x)), ";\n", type(x)	
3 y = x ♯ Affectation par adresse (référence)	y : val. 12.3456; adr. 0x3bca78
4 print "y : val. ", y, ";\n adr. ", hex(id(y))	
5 x = -65	x : val. -65; adr. 0x3ac708 <type 'int'> y : val. 12.3456; adr. 0x3bca78
6 print "x : val. ", x, ";\n adr. ", hex(id(x)), ";\n", type(x)	
7 print "y : val. ", y, ";\n adr. ", hex(id(y))	

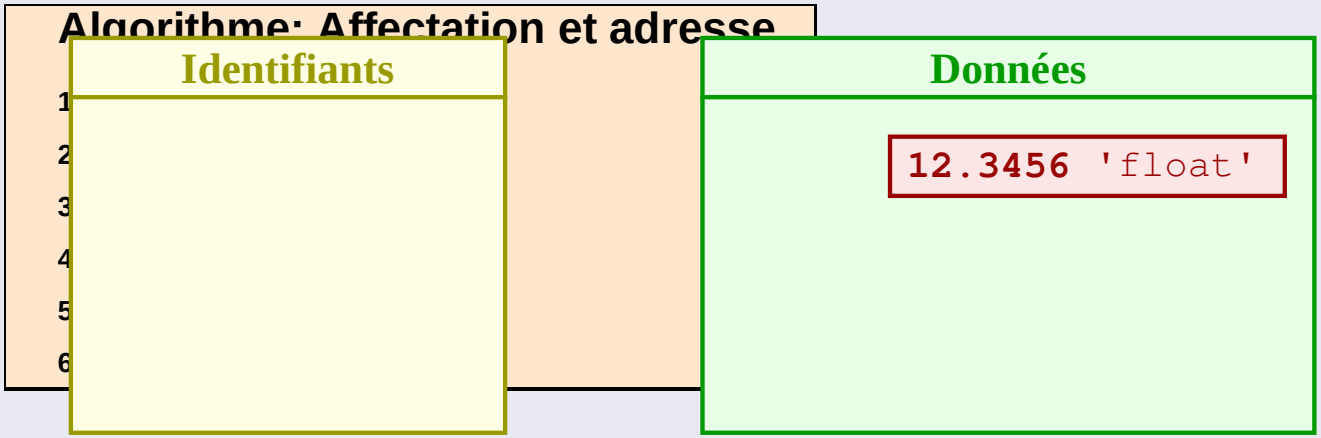
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

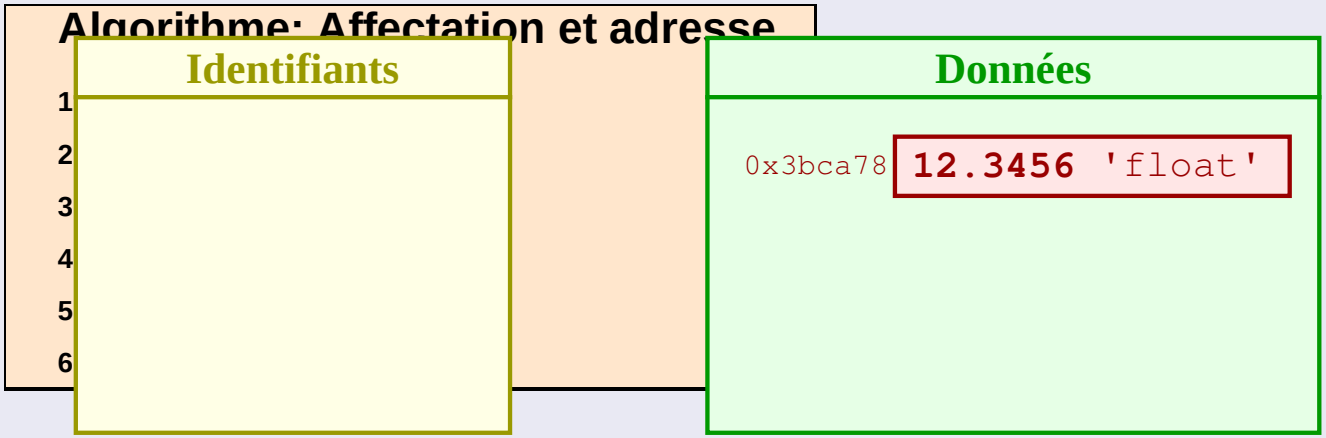
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

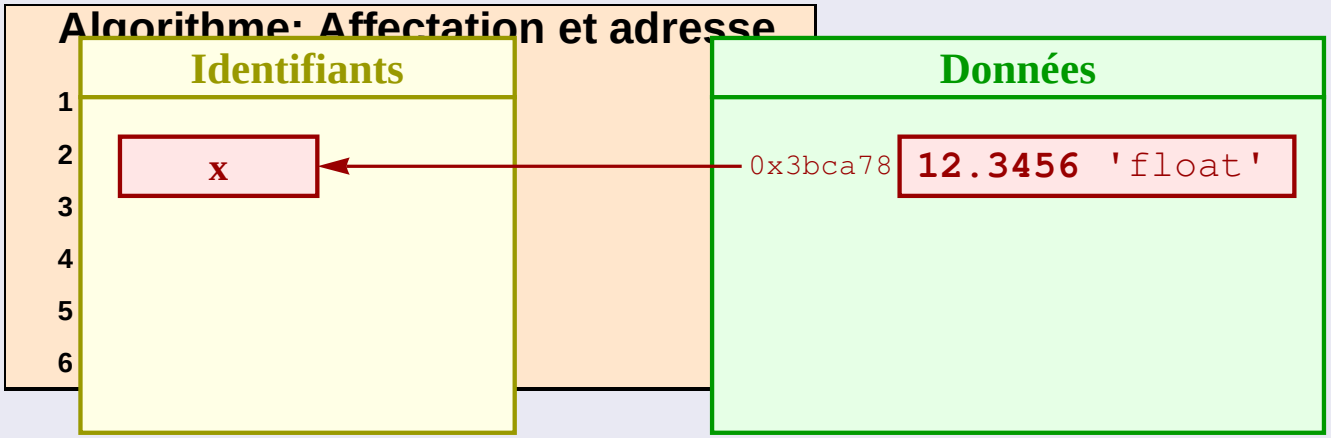
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

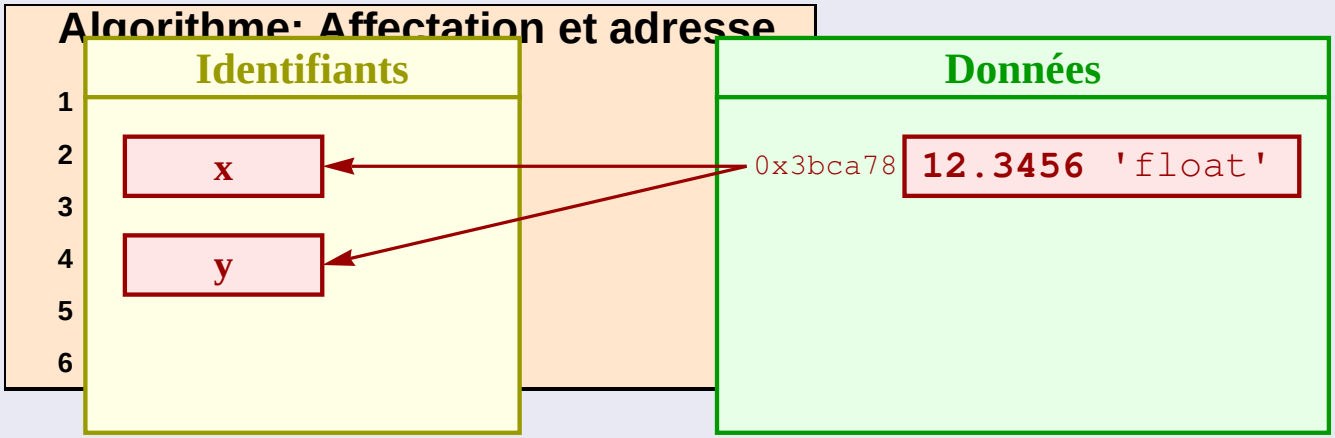
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

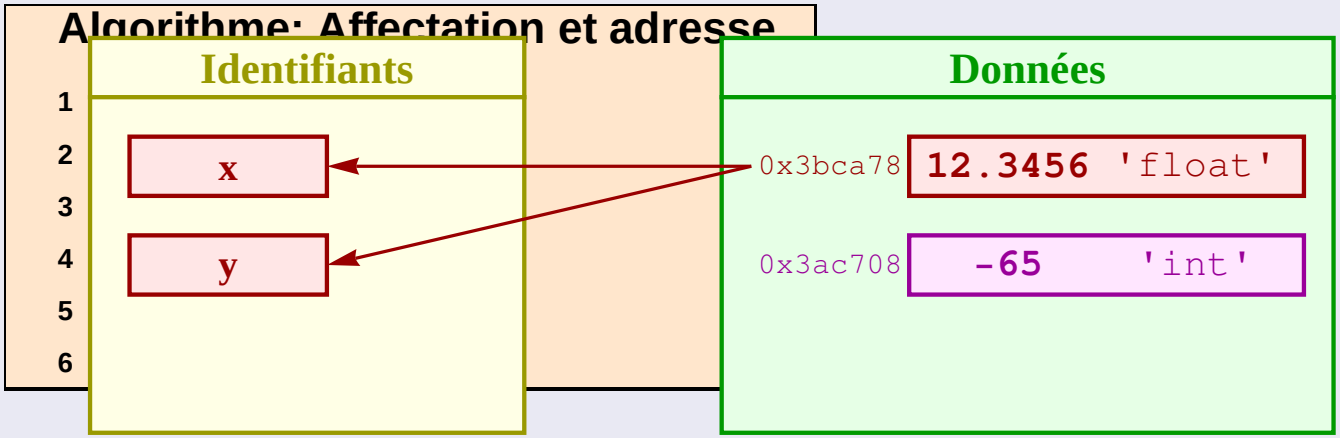
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

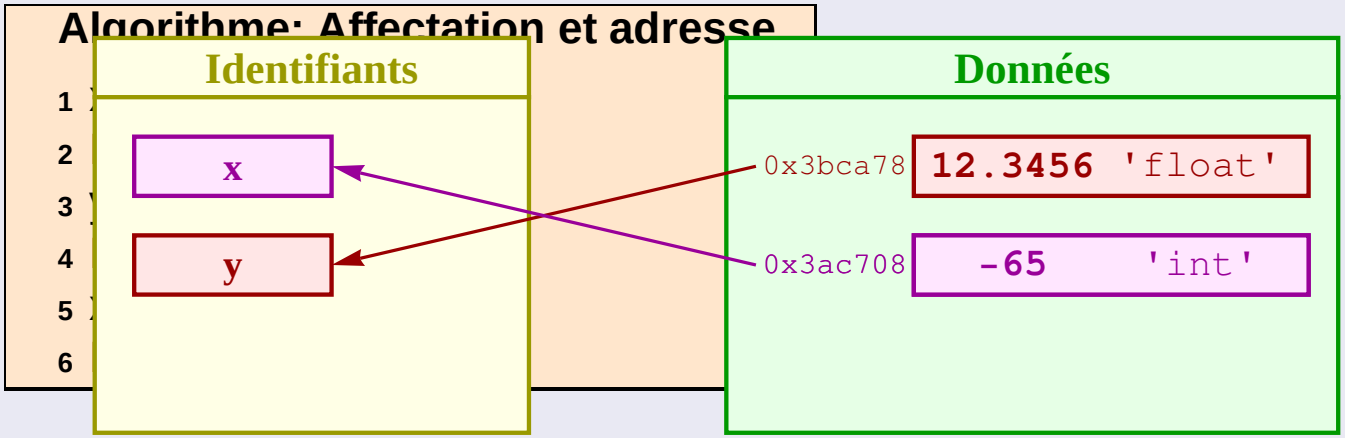
Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

Typage statique / dynamique : l'influence sur l'affectation



Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)
1	x = 12.3456	
2	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. 12.3456 ; adr. 0x3bca78 <type 'float'>
3	y = x ♯ Affectation par adresse (référence)	
4	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78
5	x = -65	
6	print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n", type(x)	x : val. -65 ; adr. 0x3ac708 <type 'int'>
7	print "y : val. ", y, " ;\n adr. ", hex(id(y))	y : val. 12.3456 ; adr. 0x3bca78

Typage statique / dynamique : incrémentation d'un entier

Algorithme: Incrémentation

1 x ← 12
2 Print x, adresse de x
3 x ← x+1
4 Print x, adresse de x

Typage statique, exemple du C++

	Instructions	Sortie (Console)
1	`int x = 12 ;`	`x : val. 12 ;` `adr. 0x28ff04`
2	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	
3	`x++ ; // Incrémentation`	`x : val. 13 ;` `adr. 0x28ff04`
4	`cout << "x : val. " << x << " ;\n adr. " << &x << endl ;`	

Typage dynamique, exemple de *Python*™

	Instructions	Sortie (Console)

1	`x = 12`	`x : val. 12 ;`
		`adr. 0x4dc984`
		`<type 'int'>`

2	`print "x : val. ", x, " ;\n adr. ", hex(id(x)), " ;\n ", type(x)`	
3	`x += 1 # Incrémentation`	`x : val. 13 ;` `adr. 0x4dc978`
4	`print "x : val. ", x, " ;\n adr. ", hex(id(x))`	

Listes

Algorithme: Création et utilisation d'une liste

1 L ← liste des carrés des 12 premiers entiers
2 Print taille de la liste
3 Print septième élément de L
4 Print la liste des trois premiers éléments de L

Liste en *Mathematica*®

	Instructions	Sortie (Console)
1	`L=Table[i^2,{i,12}];(*L=♯^2&/@Range[12]*)`	
2	`Length[L]`	`12`
3	`L[[7]] (*ou Part[L,7]*)`	`49`
4	`Take[L,3]`	`{1,4,9}`

Liste en *Python*™ : le type 'list'

	Instructions	Sortie (Console)
1	`L=[ ] ♯ liste vide, ou L = [ i**2 for i in range(1,13) ]`	
2	`for i in range(1,13) : L.append(i**2)`	
3	`print len(L), type(L)`	`12 <type 'list'>`
4	`print L[6] ♯ la numérotation commence à zéro`	`49`
5	`print L[ :3]`	`[1, 4, 9]`

Operations sur les listes

Opération	*Python*™	*Mathematica*®
Ajout de l'élément e à la fin de la liste L	L.append(e)	L=Append[L,e]
Coller la liste L2 à la fin de la liste L1	L1.extend(L2) (adresse inchangée) L1 = L1 + L2 (adresse modifiée)	L1=Join[L1,L2]
Insertion de l'élément e au i-ème rang de L	L.insert($i-1$,e)	L=Insert[L,e, i]
Enlever les éléments du rang i au rang j	del L[$i-1$: j]	L=Drop[L,{ i , j }]

Pour en savoir plus :

- http://docs.python.org/tutorial/datastructures.html
- http://reference.wolfram.com/mathematica/guide/ListManipulation.html

Tableau à deux indices

Liste de listes

- *Mathematica*® : M = {{1,2,3}, {4,5,6}} ;
- *Python*™ : M = [[1,2,3], [4,5,6]] ♯ (type 'list')

Bibliothèques spécifiques

- *Python*™ :

```
♯ pour travailler sur les tableaux
from numpy import array
M=array( [ [1,2,3], [4,5,6] ] ) ♯ (type 'numpy.ndarray')
```

Pour en savoir plus :

- Python'ndarray : http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

Chaînes de caractères

Algorithme: Création et manipulation d'une chaîne de caractères

```
1 L ← "abcdefghij"
2 Print taille de la chaîne
3 Print septième caractère de L
4 Print la liste des trois premiers caractères de L
```

Chaîne de caractères en *Mathematica*®

	Instructions	Sortie (Console)
1	`L="abcdefghij";`	
2	`StringLength[L]`	`10`
3	`StringTake[L,{7}]`	`g`
4	`StringTake[L,3]`	`abc`

Chaîne en *Python*™ : le type 'str', qui se manipule comme une liste

	Instructions	Sortie (Console)
1	`L="abcdefghij"` ♯ ou L='abcdefghij'	
2	`print len(L), type(L)`	`10 <type 'str'>`
3	`print L[6]` ♯ la numérotation commence à zéro	`g`
4	`print L[ :3]`	`abc`

Operations sur les chaînes de caractères

Opération	*Python*™	*Mathematica*®
Obtenir le nombre de caractères de C	len(C)	StringLength[C]
Coller la chaine C2 à la fin de la chaîne C1	C1=C1+C2	C1=StringJoin[C1,C2] C1=C1<>C2
Insertion de la chaine C2 au i-ème rang de C1	C1=C1[: $i-1$]+C2+C1[$i-1$:]	C1=StringInsert[C1,C2, i]
Enlever les éléments du rang i au rang j	C1=C1[: $i-1$]+C1[j :]	C1=StringDrop[C1,{ i , j }]
Conserver les éléments du rang i au rang j	C1=C1[$i-1$: j]	C1=StringTake[C1,{ i , j }]
Ne conserver que les i premiers caractères	C1=C1[: i]	C1=StringTake[C1, i]
Enlever les i derniers caractères	C1=C1[: $-i$]	C1=StringDrop[C1, $-i$]
Prélever les i derniers caractères	C2 = C1[$-i$:]	C2 = StringTake[C1, $-i$]

Pour en savoir plus :

- http://docs.python.org/tutorial/introduction.html#strings
- http://reference.wolfram.com/mathematica/tutorial/OperationsOnStrings.html

Mode "console" – Command Line Interface

- Affichage

Python™
print ’valeur de x : ’, x

- Saisie

Python™
x = raw_input (’Saisir la valeur de x : ’)

Interface graphique – Graphical User Interface

- Création de fenêtre(s) de dialogue contenant des ’widgets’ (boutons, case à cocher, zone de texte, canevas, . . .)
- Appel à une bibliothèque spécialisée multiplateforme (Qt, GTK, Tk, wxWidgets, . . .)

Mode "console" – Command Line Interface

- Affichage

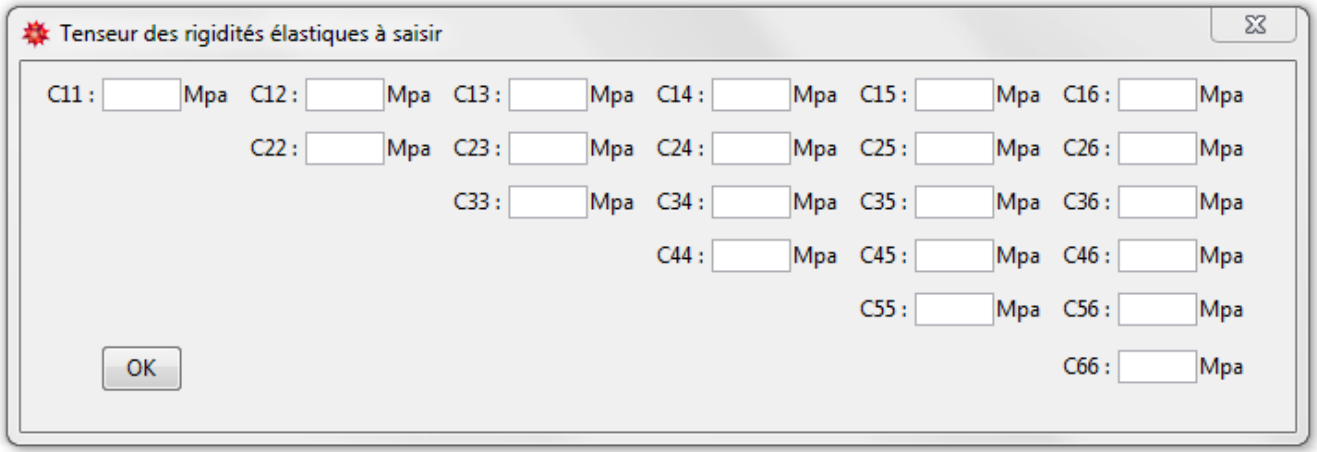
Python™
print ’valeur de x : ’, x

- Saisie

Python™
x = raw_input (’Saisir la valeur de x : ’)

Interface graphique – Graphical User Interface

- Création de fenêtre(s) de dialogue contenant des ’widgets’ (boutons, case à cocher, zone de texte, canevas, . . .)
- Appel à une bibliothèque spécialisée multiplateforme (Qt, GTK, Tk, wxWidgets, . . .)



Fichiers

- ouverture
- lecture/écriture
- fermeture

Ouverture-lecture-fermeture en *Mathematica*®

Instructions	Sortie (Console)

`f = OpenRead["data.txt"]`
`l = ReadList[f, String];`
 	
`InputForm[l]`
 	
`Close[f]`
	`InputStream["data.txt", 15]`
	
	`{"u[mV]\ti[mA]","250.1\t2.32","200.5\t2.61",`
	`"212.4\t2.45","270.0\t1.89","231.1\t2.42",`
	`"222.2\t2.44","264.8\t2.00","198.6\t2.70"}`

Ouverture-lecture-fermeture en *Python*™

Instructions	Sortie (Console)

`f = open("data.txt", "r")`
`print type(f)`
`l = [e for e in f]`
 	
`print l`
 	
`f.close()`
	`<type 'file'>`
	
	`['u[mV]\ti[mA]\n','250.1\t2.32\n','200.5\t2.61\n',`
	`'212.4\t2.45\n','270.0\t1.89\n','231.1\t2.42\n',`
	`'222.2\t2.44\n','264.8\t2.00\n','198.6\t2.70']`

if,then,else

Algorithme: structure d'un if/then/else

```
1 if expression then
2       faire action 1
3 else
4       faire action 2
  (fin)
```

Algorithme: if/then/else imbriqués

```
1  if expression_1 then
2        faire action 1
3  else
4        if expression_2 then
5              faire action 2
6        else
7              if expression_3 then
8                    faire action 3
9              else
10                   ...
   (fin)
```

cf. http://docs.python.org/2/reference/compound_stmts.html#the-if-statement

if,then,else

Algorithme: structure d'un if/then/else

```
1 if expression then
2       faire action 1
3 else
4       faire action 2
  (fin)
```

Algorithme: if/then/else imbriqués

```
 1 if expression_1 then
 2       faire action 1
 3 else
 4       if expression_2 then
 5             faire action 2
 6       else
 7             if expression_3 then
 8                   faire action 3
 9             else
10                   ...
   (fin)
```

Algorithme: structure d'un if/then/else

```
1 if expression_1 then
2       faire action 1
3 else if expression_2 then
4       faire action 2
5 else if expression_3 then
6       faire action 3
7       ...
  (fin)
```

cf. http://docs.python.org/2/reference/compound_stmts.html#the-if-statement

if,then,else

Mathematica®

Instructions

```
1  If[expression1,
2     action1,
3     If[expression2,
4        action2,
5        If[expression3,
6           action3,
7           ...
8        ]
9     ]
10 ]
```

Instructions

```
1 Witch[
2    expression1, action1,
3    expression2, action2,
4    expression3, action3,
5    True, (* default *)...
6 ]
```

Voir aussi **Switch**

Python™

Instructions

```
1  if expression_1 :
2      action_1
3  else :
4      if expression_2 :
5          action_2
6      else :
7          if expression_3 :
8              action_3
9          else :
10             ...
```

Instructions

```
1 if expression_1 :
2     action_1
3 elif expression_2 :
4     action_2
5 elif expression_3 :
6     action_3
7 else :
8     ...
```

http://docs.python.org/tutorial/controlflow.html

Boucle for

Algorithme: boucle inconditionnelle

```
1 for i ← b to e step s
2     action(s)
```

Mathematica®

Instructions

```
1 Table[
2     action(s),
3     {i,b,e,s}
4 ]
```

Instructions

```
1 For[
2     i=b,i<=e, i+=s,
3     action(s)
4 ]
```

Python™

Instructions

```
1 # b, e et s entiers
2 for i in range(b,e+s,s) :
3     action(s)
```

Instructions

```
1 for x in liste :
2     action(s)
```

While : exemple d'utilisation

Algorithme: Tirage pseudo-aléatoire de nb valeurs, selon une loi géométrique $\mathcal{G}(p)$

Entrée(s): p, nb
Retour(s): l

```
1  q ← 1 − p
2  l ← liste vide
3  for i ← 1 to nb
4      r ← valeur pseudo-aléatoire tirée avec une loi uniforme sur [0, 1]
5      n ← 1
6      qn ← q
7      while r < qn
8          Incrémenter n (de 1)
9          qn ← qn ∗ q
10     Rajouter n à la fin de la liste l
11 return liste l
```

While : exemple d'utilisation

Écriture en *Python*™

Instructions

```
1  from numpy.random import uniform
2  p = float(raw_input("Parametre p ? "))
3  nb = int(raw_input("Nombre de valeurs ? "))
4  q = 1 - p
5  l = []
6  for i in range(nb) :
7      r = uniform()
8      n = 1
9      qn = q
10     while r < qn :
11         n += 1
12         qn *= q
13     l.append(n)
14 print l
```

Exemples de sorties pour $p = 0.02$ et $nb = 9$:

[123, 37, 2, 28, 24, 177, 39, 29, 2]
[51, 299, 37, 63, 41, 45, 57, 12, 82]
etc.

Écriture en *Mathematica*®

Instructions

```
1  l = With[{p=0.02,nb=20},
2     q = 1 - p;
3     Table[
4        r = RandomReal[];
5        n = 1;
6        qn = q;
7        While[ r < qn ,
8           n++;
9           qn *= q
10       ];
11       n,
12       {nb}
13    ]
14 ]
```

Exemples pour $p = 0.02$ et $nb = 9$:

{49, 41, 13, 67, 39, 100, 2, 164, 21}
{26, 3, 49, 163, 67, 3, 72, 13, 22}
etc.

Interruption(s) dans une boucle : break et continue

Interruption(s) par 'break'

Algorithme: Interruptions multiples

while expression
action(s) n°1
if expression1 **then**
    break (sortie de boucle)
action(s) n°2
if expression2 **then**
    continue (itération suivante)
action(s) n°3
if expression3 **then**
    break
...
suite (après sortie de boucle) ...

Interruption(s) dans une boucle : break et continue

Interruption(s) par 'break'

Algorithme: Interruptions multiples

while expression
action(s) n°1
if expression1 **then**
    break (sortie de boucle)
action(s) n°2
if expression2 **then**
    continue (itération suivante)
action(s) n°3
if expression3 **then**
    break
. . .
suite (après sortie de boucle) . . .

Cas particulier : boucle principale pour interface graphique

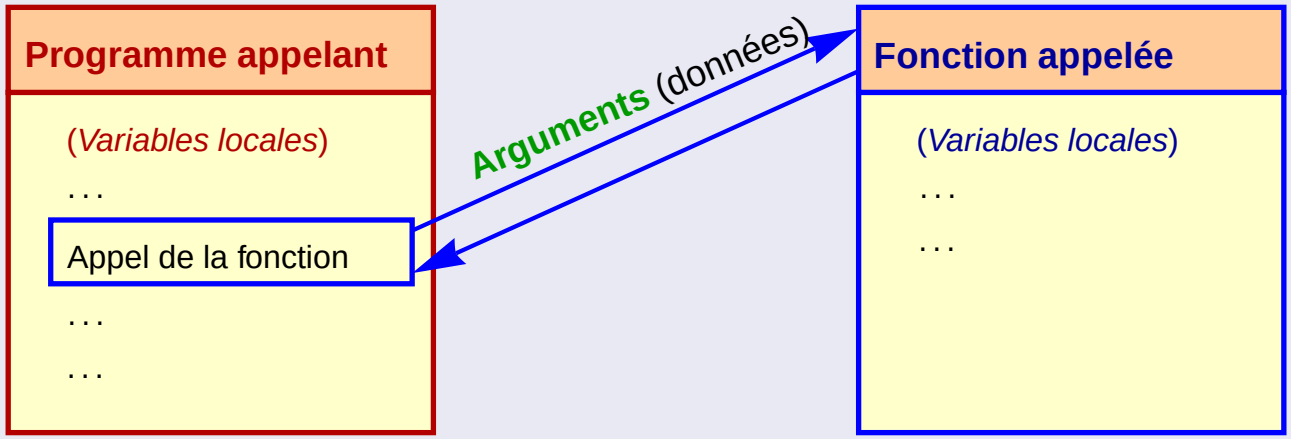
Algorithme: 'mainloop' avec un GUI

(* begin main *)
(initialisations, instructions de création et d'ouverture)
(* mainloop : veille des évènements souris et clavier *)
while true
action(s) n°1
    if expression1 **then break**
    action(s) n°2
    if expression2 **then continue**
    action(s) n°3
    if expression3 **then break**
    . . .
(instructions de fermeture et de suppression)
(* end main *)

Approche modulaire : les Fonctions

- Tous les paradigmes de programmation (impérative, fonctionnelle, orientée objet) encouragent la programmation modulaire : **décomposer un problème complexe en problèmes élémentaires** plus faciles à analyser
- Chacun des problèmes élémentaires constitue une unité de programmation qui peut recevoir et retourner des données
- On parle en général de **fonction**, **procédure**, **méthode** ...
- Dans certains langages (Fortran, VB) on distingue :
  fonction $\rightarrow$ fournit des données en retour
  procédure $\rightarrow$ ne retourne rien.
- **Arguments** : les données transmises à l'appel des fonctions
- Deux mécanismes de passage des arguments aux fonctions : par **valeur** ou par **adresse**
- Le **retour** peut aussi se faire par valeur ou par adresse
- Le passage/retour par valeur peut coûter cher en mémoire si les données sont volumineuses (matrice...)

passage des arguments aux fonctions



passage par valeur

- la fonction reçoit la **valeur** des arguments passés
- la fonction ne peut pas modifier cette valeur
- mode de passage par défaut en VB, C, C++

passage par référence

- la fonction reçoit l'**adresse** des arguments passés
- elle peut modifier le contenu de la mémoire à cette adresse
- mode de passage par défaut en Fortran, *Python*™
- VB : utilisation du mot-clef ByRef (ByVal par défaut)
- C++ : 2 syntaxes → **pointeur** ou **référence**

Exemple de base : entrée(s) en argument(s) / sortie en retour

Passage par valeur (langage à typage statique en général)

	Instructions	Sortie (Console)
1	//============ Fonction ============	
2	float **chgtbase1**(float **xin**, float ain, float bin)	
3	float **xout** // Déclaration des **variables locales**	
4	Print("xin : val.", **xin**, " ; \n\t adr.", Adresse(**xin**))	
5	**xout** = ain***xin** + bin	
6	Print("xout : val.", **xout**, " ; \n\t adr.", Adresse(**xout**))	
7	Return **xout**	
8	//============ Main ============	
9	float **x** // Déclaration des **variables**	
10	**x** = 3.	
11	Print("x : val.", **x**, " ; \n\t adr.", Adresse(**x**))	x : val. 3 ; adr. 0x28ff0c
12	**x** = **chgtbase1**(**x**, 0.5, -2.)	xin : val. 3 ; adr. 0x28fef0 xout : val. -0.5 ; adr. 0x28fedc
13	Print("x : val.", **x**, " ; \n\t adr.", Adresse(**x**))	x : val. -0.5 ; adr. 0x28ff0c

Exemple de base : entrée(s) en argument(s) / sortie en retour

Python™ (typage dynamique) : **passage par référence**

	Instructions	Sortie (Console)
1	`# -*- coding : latin-1 -*-`	
2	`#========== Fonction(s) ==========`	
3	`def adr(var) : return hex(id(var))`	
4	`def chgtbase1(xin , ain, bin) :`	
5	`    print "xin : val.", xin , ";\n adr.", adr(xin )`	
6	`    xout = ain*xin + bin`	
7	`    print "xout : val.", xout , ";\n adr.", adr(xout )`	
8	`    return xout`	
9	`#============ Main ============`	
10	`x = 3.`	
11	`print "x : val.", x, ";\n adr.", adr(x)`	`x : val. 3;` `  adr. 0x28ff0c`

12	`x = chgtbase1(x, 0.5, -2.)`	`xin : val. 3;`
		`  adr. 0x28ff0c`
		`xout : val. -0.5;`
		`  adr. 0x28fef0`

13	`print "x : val.", x, ";\n adr.", adr(x)`	`x : val. -0.5;` `  adr. 0x28fef0`

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Passage par valeur : ça coince. . .

	Instructions	Sortie (Console)
1	//============ Fonction ============	
2	void **chgtbase2**(float **xloc**, float ain, float bin)	
3	Print("xloc : val.", **xloc**, "; \n\t adr.", Adresse(**xloc**))	
4	**xloc** = ain***xloc** + bin	
5	Print("xloc : val.", **xloc**, "; \n\t adr.", Adresse(**xloc**))	
6	//======= Programme principal =======	
7	x = 3.0	
8	Print("x : val.", **x**, "; \n\t adr.", Adresse(**x**))	x : val. 3; adr. 0x28ff0c
9	**chgtbase2**(**x**, 0.5, -2.)	xloc : val. 3; adr. 0x28fef0 xloc : val. -0.5; adr. 0x28fef0
10	Print("x : val.", **x**, "; \n\t adr.", Adresse(**x**))	x : val. 3; adr. 0x28ff0c

- On modifie uniquement la copie de la valeur de *x*

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Python™ : ça coince aussi pour un **objet immuable**. . .

- **Objet immuable** (ou non-mutable) : qui ne peut être modifié.
  Les types de base (’int’, ’float’, ’str’, ’bool’) ainsi que le type ’tuple’ sont immuables ;
  http://docs.python.org/2/glossary.html#term-immutable
- **Objet mutable** : dont le contenu peut être modifié. Exemple : le type ’list’.
  http://docs.python.org/2/glossary.html#term-mutable

	Instructions	Sortie (Console)
4	def **chgtbase2**(**xloc**, ain, bin) :	
5	print "xloc : val.", **xloc**, " ;\n adr.", adr(**xloc**)	
6	**xloc** = ain***xloc** + bin	
7	print "xloc : val.", **xloc**, " ;\n adr.", adr(**xloc**)	
10	a = 3.0	
11	print "x : val.", **x**, " ;\n adr.", adr(**x**)	x : val. 3 ; adr. 0x28ff0c
12	**chgtbase2**(**x**, 0.5, -2.)	xloc : val. 3 ; adr. 0x28ff0c xloc : val. -0.5 ; adr. 0x28fef0
13	print "x : val.", **x**, " ;\n adr.", adr(**x**)	x : val. 3 ; adr. 0x28ff0c

- ligne 6 : l’affectation de l’identifiant **xloc** le fait référencer un nouvel objet (typage dynamique)

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en VB (typage statique) : le passage par référence

	Instructions	Sortie (Console)
1	`Sub chgtbase3(ByRef xloc As Double, _`	
2	`  ByVal ain As Double, ByVal bin As Double)`	
3	`  Console.WriteLine("xloc : val. " & xloc & " ;\n adr. "_`	
4	`  & VarPtr(xloc))`	
5	`  xloc = ain*xloc + bin ;`	
6	`  Console.WriteLine("xloc : val. " & xloc & " ;\n adr. "_`	
7	`  & VarPtr(xloc))`	
8	`End Sub`	
9	`Sub Main()`	
10	`  Dim x As Double`	
11	`  x = 3.`	
12	`  Console.WriteLine("x : val. " & x & " ;\n adr. " & VarPtr(x))`	`x : val. 3 ;` `  adr. 0x28ff0c`

13	`  chgtbase3(x, 0.5, -2.) ;`	`xloc : val. 3 ;`
		`  adr. 0x28ff0c`
		`xloc : val. -0.5 ;`
		`  adr. 0x28ff0c`

14	`  Console.WriteLine("x : val. " & x & " ;\n adr. " & VarPtr(x))`	`x : val. -0.5 ;` `  adr. 0x28ff0c`
15	`End Sub`	

- Voir par exemple http://msdn.microsoft.com/fr-fr/vbasic/default.aspx
- Et plus particulièrement http://msdn.microsoft.com/fr-fr/library/ddck1z30.aspx

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.

	Instructions	**Sortie (Console)**
4	`def chgtbase3(box, ain, bin) :`	
5	`    print "box : val.", box, " ;\n adr.", adr(box)`	
6	`    box[0] *= ain`	
7	`    box[0] += bin`	
8	`    print "box : val.", box, " ;\n adr.", adr(box)`	
9	`#============ Main ============`	
10	`x = 3.`	`x : val. 3.0 ;`
11	`print "x : val.", x, " ;\n adr.", adr(x)`	`  adr. 0x52c5b0`
12	`boite = [x] # On met x dans la boîte`	`boite : val. [3.0] ;`
13	`print "boite : val.", boite, " ;\n adr.", adr(boite)`	`  adr. 0x263e260`

14	`chgtbase3(boite, 0.5, -2.)`	`box : val. [3.0] ;`
		`  adr. 0x263e260`
		`box : val. [-0.5] ;`
		`  adr. 0x263e260`

15	`print "boite : val.", boite, " ;\n adr.", adr(boite)`	`boite : val. [-0.5] ;` `  adr. 0x263e260`
16	`x = boite[0] # On ressort x de la boîte`	`x : val. -0.5 ;`
17	`print "x : val.", x, " ;\n adr.", adr(x)`	`  adr. 0x52c580`

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.

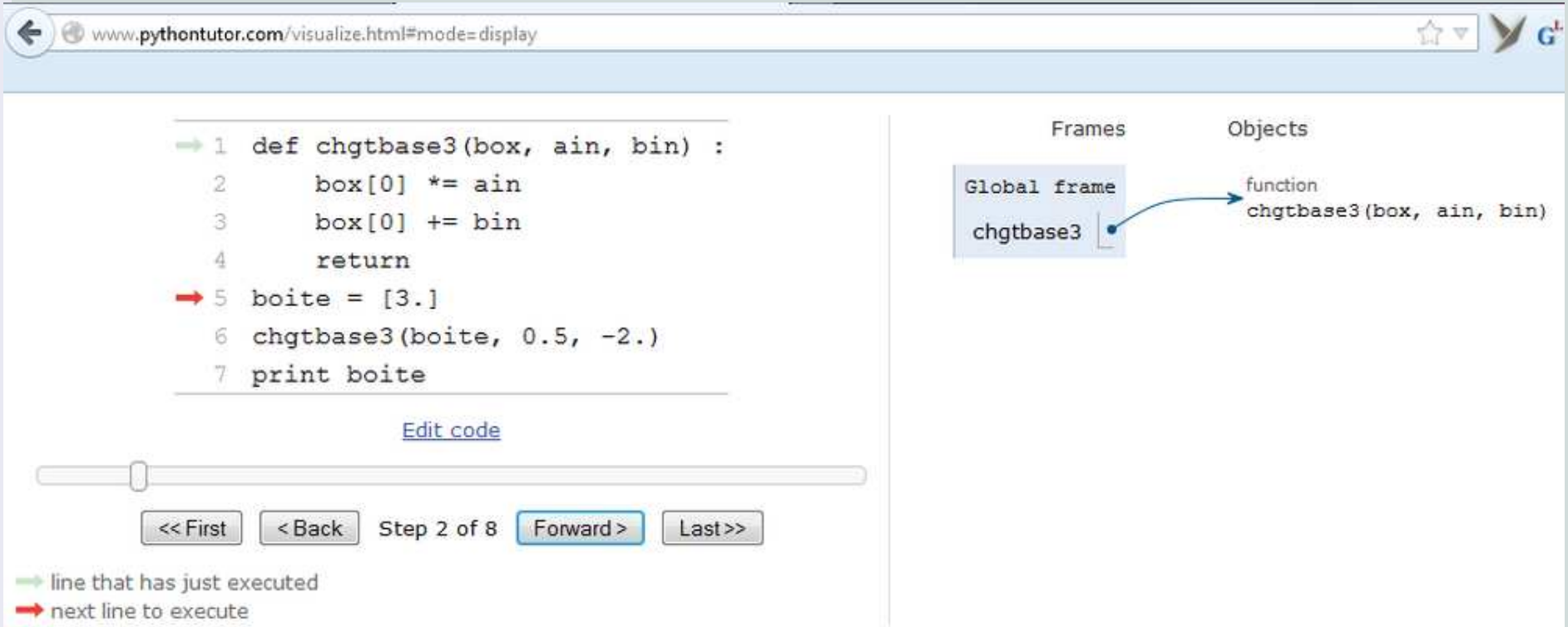

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.




```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.

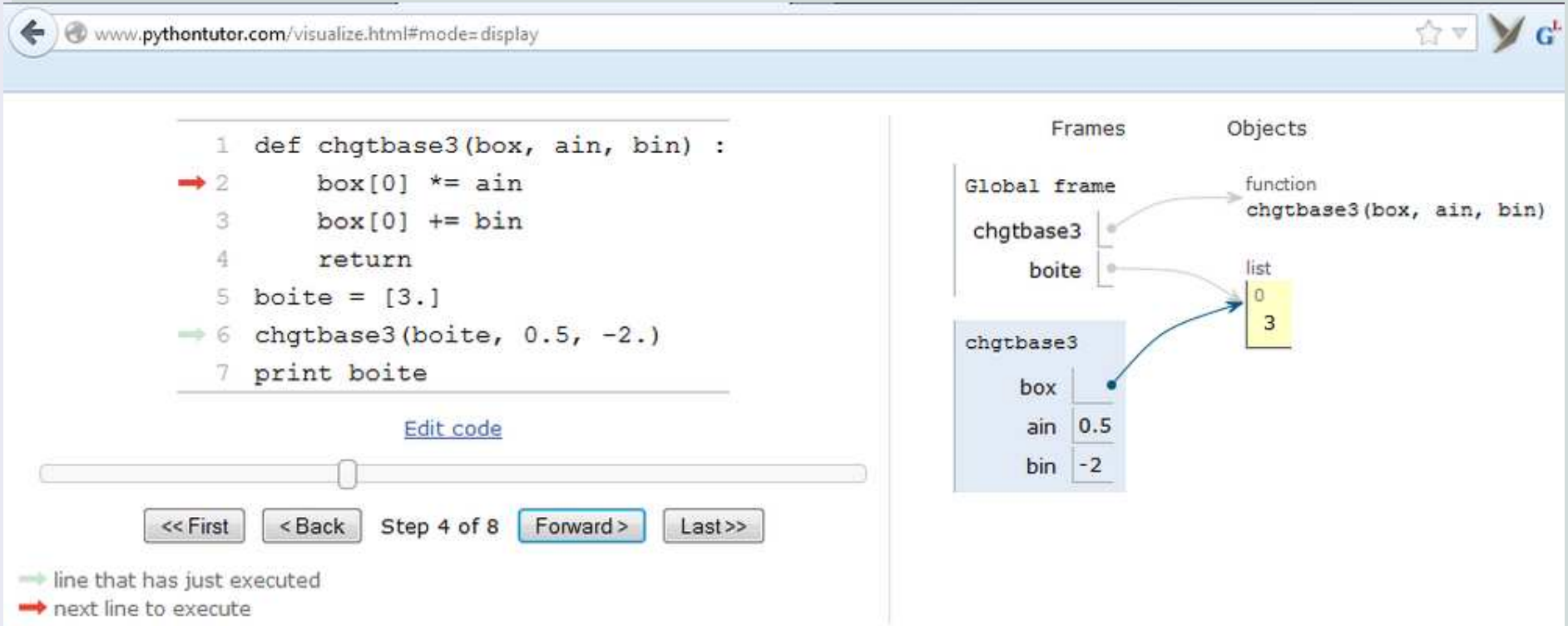

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.



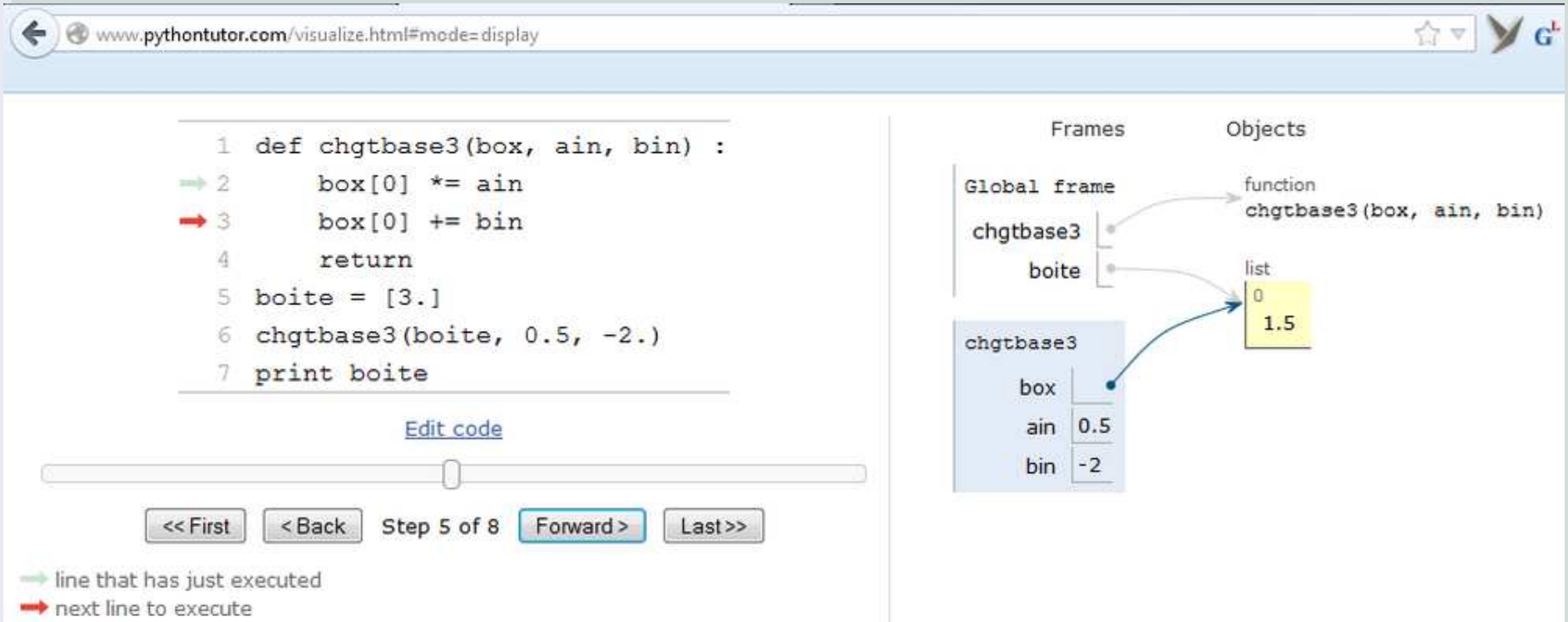

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.



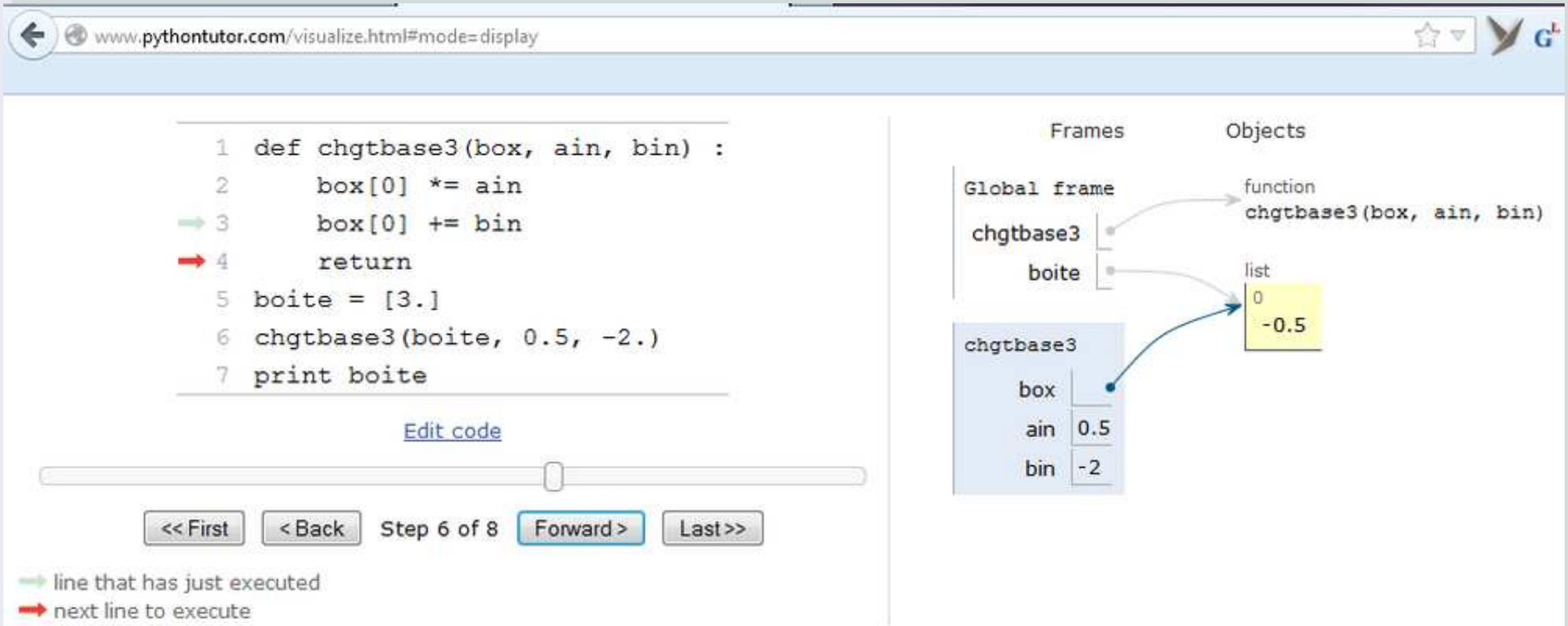

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.



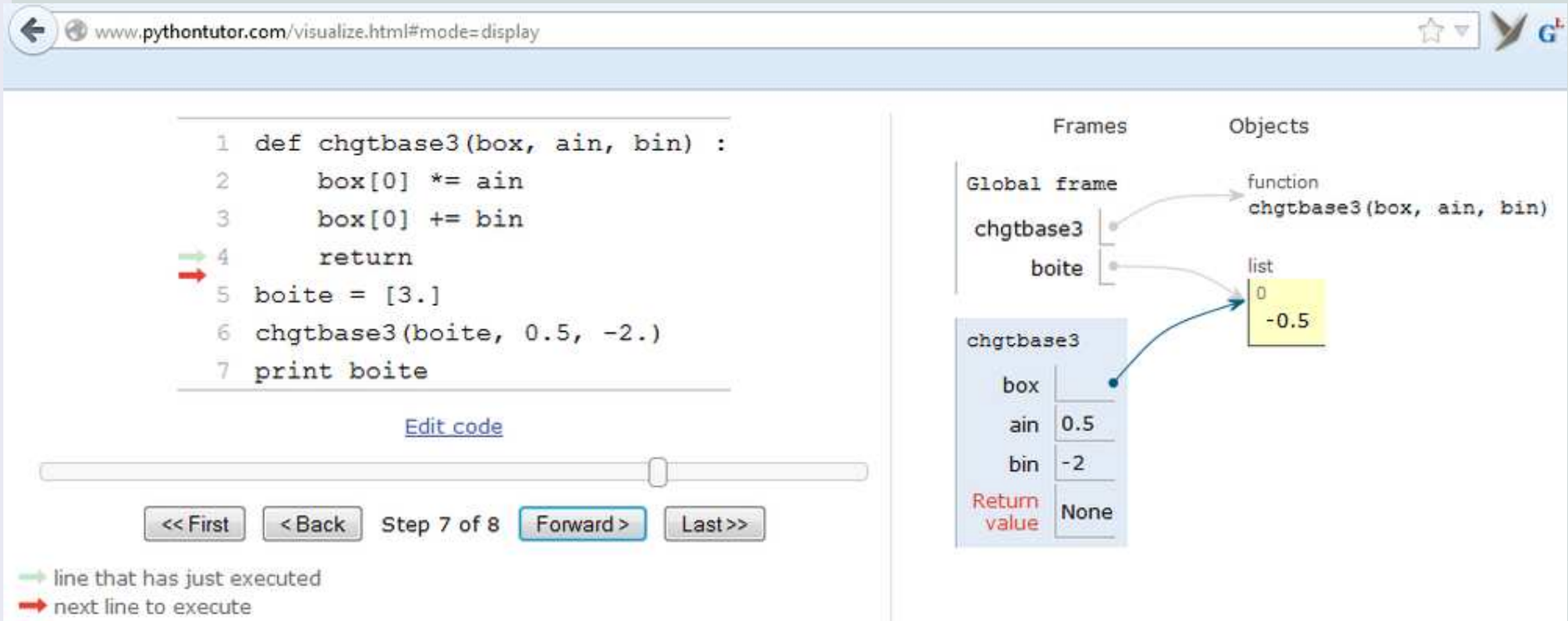

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5] ;
  adr. 0x263e260
x : val. -0.5 ;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.



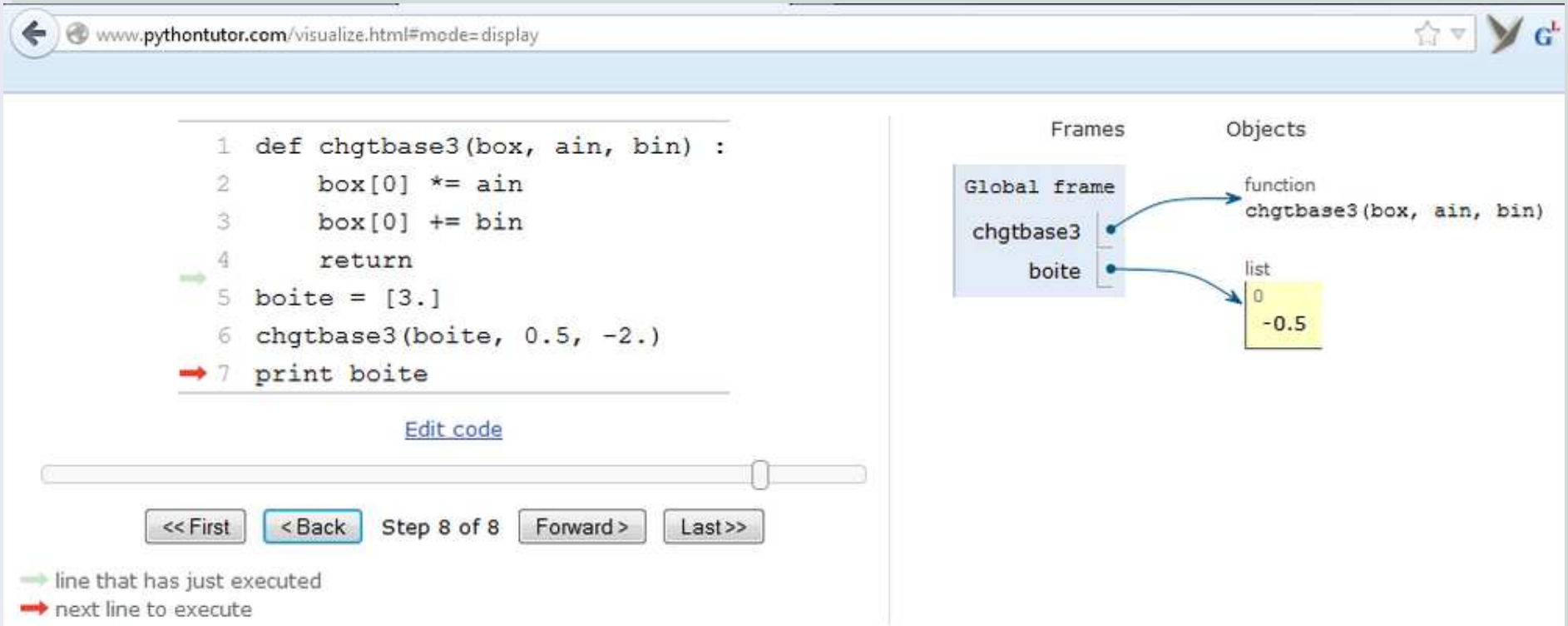

```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.




```
15 print "boite : val.", boite, " ;\n adr.", adr(boite)
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, " ;\n adr.", adr(x)
```

```
boite : val. [-0.5];
  adr. 0x263e260
x : val. -0.5;
  adr. 0x52c580
```

Exemple avancé : entrée(s) et sortie(s) en argument(s)

Solution en *Python*™ : on utilise un **objet mutable**

- Ici, l'objet mutable est une liste dont on modifie le contenu.


```
16 x = boite[0] # On ressort x de la boîte
17 print "x : val.", x, ";\n adr.", adr(x)
```

```
x : val. -0.5;
  adr. 0x52c580
```

Récursivité

- Fonction qui s'appelle elle-même
- Necessité d'un test d'arrêt
- Phénomène d'empilement des noms et des valeurs $\Longrightarrow$ risque de saturation de la mémoire
- Exemples : *cf.* algorithmes ci-après.

```
def maFonction( <arguments> ) :
    ...
    (calcul de argloc)
    ...
    if continuer then :
        ...
        maFonction( <argloc> )
        ...
    # end if
    ...
# end def
```

Calcul de la valeur d'une fonction polynomiale

Objectif : calculer $\sum_{k=0}^{n} a_k\, x^k$, en fonction de x et de la liste $coefpol = [\, a_0\,,\, a_1\,,\, \cdots,\, a_n\,]$

Algorithme: n add., n mult., n exp.

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[0]
3 for k ← 1 to deg
4     valeur ← valeur+coefpol[k]*x^k
5 return valeur
```

Calcul de la valeur d'une fonction polynomiale

Objectif : calculer $\sum_{k=0}^{n} a_k\, x^k$, en fonction de x et de la liste $coefpol = [\, a_0\,,\, a_1\,,\, \cdots,\, a_n\,]$

Algorithme: n **add.,** n **mult.,** n **exp.**

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[0]
3 for k ← 1 to deg
4     valeur ← valeur+coefpol[k]*x^k
5 return valeur
```

Algorithme: n **add.,** $2n$ **mult.**

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[0]
3 xpuisk ← 1
4 for k ← 1 to deg
5     xpuisk ← xpuisk * x
6     valeur ← valeur+coefpol[k]*xpuisk
7 return valeur
```

Calcul de la valeur d'une fonction polynomiale

Objectif : calculer $\sum_{k=0}^{n} a_k\, x^k$, en fonction de x et de la liste $coefpol = [\, a_0\,,\, a_1\,,\, \cdots,\, a_n\,]$

Algorithme: n **add.,** n **mult.,** n **exp.**

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[0]
3 for k ← 1 to deg
4     valeur ← valeur+coefpol[k]*x^k
5 return valeur
```

Algorithme: n **add.,** $2n$ **mult.**

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[0]
3 xpuisk ← 1
4 for k ← 1 to deg
5     xpuisk ← xpuisk * x
6     valeur ← valeur+coefpol[k]*xpuisk
7 return valeur
```

Algorithme: **Horner** n **add.,** n **mult.**

Entrée(s): x, *coefpol*
Retour(s): *valeur*

```
1 deg ← (taille de la liste coefpol) - 1
2 valeur ← coefpol[n]
3 for k ← deg − 1 to 0 step −1
4     valeur ← coefpol[k] + x * valeur
5 return valeur
```

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	
-1	–	–	–	–	–	–	0

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	
-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	

-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	

-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	

-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2
3	0	1	4	-1	–	–	2

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	
-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2
3	0	1	4	-1	–	–	2
3	5	1	4	-1	0	–	2

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	

-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2
3	0	1	4	-1	–	–	2
3	5	1	4	-1	0	–	2
3	5	1	4	-1	6	0	2

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	
-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2
3	0	1	4	-1	–	–	2
3	5	1	4	-1	0	–	2
3	5	1	4	-1	6	0	2
1	3	10	12	12	45	53	

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Tri simple par liste chaînée

Algorithme: Tri simple

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  n ← length(L)
2  suivant ← liste de (n + 1) zéros
3  suivant[0] ← −1
4  for i ← 1 to n − 1
5      prec ← n
6      j ← suivant[n]  ♯ numéro du plus petit
7      while j ≠ −1 and L[j] < L[i]
8          prec ← j
9          j ← suivant[j]
10     suivant[prec] ← i
11     suivant[i] ← j  ♯ insertion du n°i
12 Ltriee ← liste vide
13 i ← suivant[n]
14 while i ≠ −1
15     Ajouter L[i] à la fin de Ltriee
16     i ← suivant[i]
17 return Ltriee
```

0	1	2	3	4	5	6	7
12	3	1	45	53	10	12	
-1	–	–	–	–	–	–	0
-1	0	–	–	–	–	–	1
-1	0	1	–	–	–	–	2
3	0	1	-1	–	–	–	2
3	0	1	4	-1	–	–	2
3	5	1	4	-1	0	–	2
3	5	1	4	-1	6	0	2
1	3	10	12	12	45	53	

Nombre moyen de tests :

$$\mathbb{E}_1 = 0\,;\ \mathbb{E}_n = \mathbb{E}_{n-1} + \frac{n+1}{2} - \frac{1}{n}$$

$$\text{d'où } \mathbb{E}_n = \frac{n(n+3)}{4} - \sum_{k=1}^{n} \frac{1}{k}$$

$$= \frac{n(n+3)}{4} - \ln(n) - \gamma + \mathcal{O}\left(\frac{1}{n}\right)$$

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Quicksort (Tri rapide)

Algorithme: quicksort (récursif)

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  a ← liste[0]
2  deb ← 1
3  fin ← (nombre d'éléments de liste)−1
4  while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
5  while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
6  while deb ⩽ fin
7      permuter liste[deb] et liste[fin]
8      while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
9      while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
10 Linf ← liste[1 : debut]
11 if (nombre d'éléments de Linf)> 1 : Linf ←quicksort(Linf)
12 Lsup ← liste[debut :]
13 if (nombre d'éléments de Lsup)> 1 : Lsup ←quicksort(Lsup)
14 Ltriee ← listes Linf, [a], Lsup mises bout à bout
15 return Ltriee
```

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Quicksort (Tri rapide)

Algorithme: quicksort (récursif)

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  a ← liste[0]
2  deb ← 1
3  fin ← (nombre d'éléments de liste)−1
4  while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
5  while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
6  while deb ⩽ fin
7      permuter liste[deb] et liste[fin]
8      while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
9      while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
10 Linf ← liste[1 : debut]
11 if (nombre d'éléments de Linf)> 1 : Linf ←quicksort(Linf)
12 Lsup ← liste[debut :]
13 if (nombre d'éléments de Lsup)> 1 : Lsup ←quicksort(Lsup)
14 Ltriee ← listes Linf, [a], Lsup mises bout à bout
15 return Ltriee
```

0	1	2	3	4	5	6
12	3	1	45	53	10	12
12	3	1	10	53	45	12
12	3	1	10	53	45	12

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Quicksort (Tri rapide)

Algorithme: quicksort (récursif)

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  a ← liste[0]
2  deb ← 1
3  fin ← (nombre d'éléments de liste)−1
4  while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
5  while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
6  while deb ⩽ fin
7      permuter liste[deb] et liste[fin]
8      while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
9      while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
10 Linf ← liste[1 : debut]
11 if (nombre d'éléments de Linf)> 1 : Linf ←quicksort(Linf)
12 Lsup ← liste[debut :]
13 if (nombre d'éléments de Lsup)> 1 : Lsup ←quicksort(Lsup)
14 Ltriee ← listes Linf, [a], Lsup mises bout à bout
15 return Ltriee
```

0	1	2	3	4	5	6
12	3	1	45	53	10	12
12	3	1	10	53	45	12
12	3	1	10	53	45	12

0	1	2		0	1	2
3	1	10	12	53	45	12
3	1	10	12	53	45	12

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Quicksort (Tri rapide)

Algorithme: quicksort (récursif)

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  a ← liste[0]
2  deb ← 1
3  fin ← (nombre d'éléments de liste)−1
4  while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
5  while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
6  while deb ⩽ fin
7      permuter liste[deb] et liste[fin]
8      while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
9      while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
10 Linf ← liste[1 : debut]
11 if (nombre d'éléments de Linf)> 1 : Linf ←quicksort(Linf)
12 Lsup ← liste[debut :]
13 if (nombre d'éléments de Lsup)> 1 : Lsup ←quicksort(Lsup)
14 Ltriee ← listes Linf, [a], Lsup mises bout à bout
15 return Ltriee
```

0	1	2	3	4	5	6
12	3	1	45	53	10	12
12	3	1	10	53	45	12
12	3	1	10	53	45	12

0	1	2		0	1	2
3	1	10	12	53	45	12
3	1	10	12	53	45	12

0		0		0	1	
1	3	10	12	45	12	53
1	3	10	12	45	12	53

Tri d'une liste

Objectif : trier une liste L de sorte que $i \leqslant j \implies less(L[i], L[j]) == \textbf{true}$

Quicksort (Tri rapide)

Algorithme: quicksort (récursif)

Entrée(s): *L*
Retour(s): *Ltriee*

```
1  a ← liste[0]
2  deb ← 1
3  fin ← (nombre d'éléments de liste)−1
4  while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
5  while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
6  while deb ⩽ fin
7      permuter liste[deb] et liste[fin]
8      while deb ⩽ fin and liste[deb] < a : deb ← deb + 1
9      while deb ⩽ fin and liste[fin] > a : fin ← fin − 1
10 Linf ← liste[1 : debut]
11 if (nombre d'éléments de Linf)> 1 : Linf ←quicksort(Linf)
12 Lsup ← liste[debut :]
13 if (nombre d'éléments de Lsup)> 1 : Lsup ←quicksort(Lsup)
14 Ltriee ← listes Linf, [a], Lsup mises bout à bout
15 return Ltriee
```

0	1	2	3	4	5	6
12	3	1	45	53	10	12
12	3	1	10	53	45	12
12	3	1	10	53	45	12

0	1	2		0	1	2
3	1	10	12	53	45	12
3	1	10	12	53	45	12

0		0		0	1	
1	3	10	12	45	12	53
1	3	10	12	45	12	53

				0		
1	3	10	12	12	45	53
1	3	10	12	12	45	53

Comparaison des deux algorithmes de tri

- **Tri simple par liste chaînée** :

$$\mathbb{E}_n = \frac{n(n+3)}{4} - \ln(n) - \gamma + \mathcal{O}\left(\frac{1}{n}\right)$$

Comparaison des deux algorithmes de tri

- **Tri simple par liste chaînée** :

$$\mathbb{E}_n = \frac{n(n+3)}{4} - \ln(n) - \gamma + \mathcal{O}\left(\frac{1}{n}\right)$$

- **Tri rapide "quicksort"** : $\mathbb{E}_0 = \mathbb{E}_1 = 0$;

$$\begin{aligned}\mathbb{E}_n &= (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(\mathbb{E}_k + \mathbb{E}_{n-1-k}) \\ &= (n-1) + \frac{2}{n}\sum_{k=0}^{n-1}\mathbb{E}_k\end{aligned}$$

Comparaison des deux algorithmes de tri

- **Tri simple par liste chaînée** :

$$\mathbb{E}_n = \frac{n(n+3)}{4} - \ln(n) - \gamma + \mathcal{O}\left(\frac{1}{n}\right)$$

- **Tri rapide "quicksort"** : $\mathbb{E}_0 = \mathbb{E}_1 = 0$;

$$\mathbb{E}_n = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(\mathbb{E}_k + \mathbb{E}_{n-1-k})$$

$$= (n-1) + \frac{2}{n}\sum_{k=0}^{n-1}\mathbb{E}_k$$

d'où $\mathbb{E}_n = (n-1) + 2(n+1)\sum_{j=1}^{n-1}\frac{(j-1)}{(j+1)(j+2)}$

$$= 2\,n\,\ln(n) - (4-2\gamma)\,n + 2\,\ln(n) + 2\gamma + 1 + \mathcal{O}\left(\frac{1}{n}\right)$$

Comparaison des deux algorithmes de tri

- **Tri simple par liste chaînée** :

$$\mathbb{E}_n = \frac{n(n+3)}{4} - \ln(n) - \gamma + \mathcal{O}\left(\frac{1}{n}\right)$$

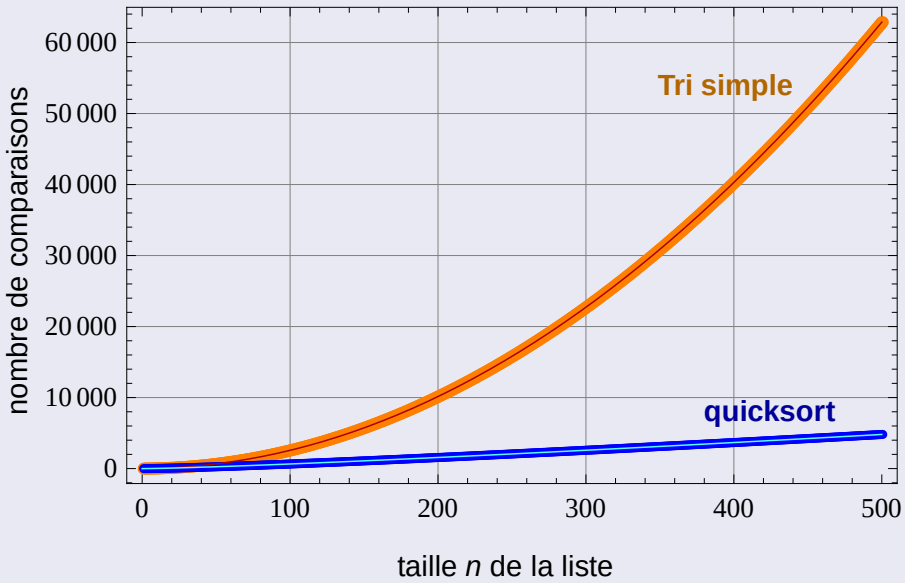
- **Tri rapide "quicksort"** : $\mathbb{E}_0 = \mathbb{E}_1 = 0$;

$$\mathbb{E}_n = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(\mathbb{E}_k + \mathbb{E}_{n-1-k})$$

$$= (n-1) + \frac{2}{n}\sum_{k=0}^{n-1}\mathbb{E}_k$$

d'où $\mathbb{E}_n = (n-1) + 2(n+1)\sum_{j=1}^{n-1}\frac{(j-1)}{(j+1)(j+2)}$

$$= 2\,n\,\ln(n) - (4-2\gamma)\,n + 2\,\ln(n) + 2\gamma + 1 + \mathcal{O}\left(\frac{1}{n}\right)$$



Transformée de Fourier discrète

Problématique : entrée $X = \begin{bmatrix} x_0 \\ x_1 \\ | \\ x_k \\ | \\ x_{n-1} \end{bmatrix}$; réponse $Y = \begin{bmatrix} y_0 \\ y_1 \\ | \\ y_k \\ | \\ y_{n-1} \end{bmatrix}$; calculer $y_j = \sum_{k=0}^{n-1} x_k \, \mathrm{e}^{-2\,\mathrm{i}\,\pi\,\frac{j\,k}{n}}$

Écriture matricielle : $Y = \mathcal{G}(X) = \begin{bmatrix} 1 & 1 & - & 1 & - & 1 \\ 1 & z & - & z^{k} & - & z^{n-1} \\ 1 & z^{2} & - & z^{2k} & - & z^{n-2} \\ | & | & - & | & - & | \\ 1 & z^{j} & - & z^{j\,k} & - & z^{n-j} \\ | & | & - & | & - & | \\ 1 & z^{n-1} & - & z^{n-k} & - & z \end{bmatrix} \cdot X$ avec $z = \mathrm{e}^{\frac{-2\,\mathrm{i}\,\pi}{n}}$, $z^{n} = 1$.

Calcul direct

Nombre d'opérations en n^{2}

Transformée de Fourier discrète

Problématique : entrée $X = \begin{bmatrix} x_0 \\ x_1 \\ | \\ x_k \\ | \\ x_{n-1} \end{bmatrix}$; réponse $Y = \begin{bmatrix} y_0 \\ y_1 \\ | \\ y_k \\ | \\ y_{n-1} \end{bmatrix}$; calculer $y_j = \sum_{k=0}^{n-1} x_k \ \mathrm{e}^{-2\,\mathrm{i}\,\pi\,\frac{j\,k}{n}}$

Écriture matricielle : $Y = \mathcal{G}(X) = \begin{bmatrix} 1 & 1 & - & 1 & - & 1 \\ 1 & z & - & z^{k} & - & z^{n-1} \\ 1 & z^{2} & - & z^{2k} & - & z^{n-2} \\ | & | & - & | & - & | \\ 1 & z^{j} & - & z^{j\,k} & - & z^{n-j} \\ | & | & - & | & - & | \\ 1 & z^{n-1} & - & z^{n-k} & - & z \end{bmatrix} \cdot X$ avec $z = \mathrm{e}^{\frac{-2\,\mathrm{i}\,\pi}{n}}$, $z^{n} = 1$.

Calcul direct

Nombre d'opérations en n^{2}

Écriture récurrente avec n pair

$$y_j = \left[\sum_{m=0}^{n/2-1} x_{2m} \ \left(z^{2}\right)^{j\,m} \right] + z^{j} \ \left[\sum_{m=0}^{n/2-1} x_{2m+1} \ \left(z^{2}\right)^{j\,m} \right]$$

D'où, puisque $z^{n/2} = -1$, $\begin{cases} Y_{haut} &= \mathcal{G}\left(X_{pair}\right) &+ \mathrm{diag}\left(1, z, z^{2}, \cdots, z^{n/2-1}\right) \cdot \mathcal{G}\left(X_{impair}\right) \\ Y_{bas} &= \mathcal{G}\left(X_{pair}\right) &- \mathrm{diag}\left(1, z, z^{2}, \cdots, z^{n/2-1}\right) \cdot \mathcal{G}\left(X_{impair}\right) \end{cases}$

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

```
1 z ← exp(−2 i π / length(X))
2 Lz ← liste des z^k pour k variant de 0 à length(X)/2 − 1
3 return FFTrec(X, Lz)
```

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

1 $z \leftarrow \exp(-2\,\mathbb{i}\,\pi\,/\,length(X))$
2 $Lz \leftarrow$ liste des z^k pour k variant de 0 à $length(X)/2 - 1$
3 **return** **FFTrec**(X, Lz)

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

FFT

- Coût unitaire $C_1 = 2$

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

```
1 z ← exp(−2 i π / length(X))
2 Lz ← liste des z^k pour k variant de 0 à length(X)/2 − 1
3 return FFTrec(X, Lz)
```

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

FFT

- Coût unitaire $C_1 = 2$
- $C_p = 2\,C_{p-1} + 3 * 2^{p-1}$

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

```
1 z ← exp(−2 i π / length(X))
2 Lz ← liste des z^k pour k variant de 0 à length(X)/2 − 1
3 return FFTrec(X, Lz)
```

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

FFT

- Coût unitaire $C_1 = 2$
- $C_p = 2\,C_{p-1} + 3 * 2^{p-1}$
- $C_p = \dfrac{3\,p - 1}{2}\,2^p$

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

```
1  z ← exp(−2 i π / length(X))
2  Lz ← liste des z^k pour k variant de 0 à length(X)/2 − 1
3  return FFTrec(X, Lz)
```

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

FFT

- Coût unitaire $C_1 = 2$
- $C_p = 2\,C_{p-1} + 3 * 2^{p-1}$
- $C_p = \dfrac{3\,p-1}{2}\,2^p$
- $C_p = \dfrac{3\,\log_2(n)-1}{2}\,n$

Transformée de Fourier discrète

Transformée de Fourier rapide pour $n = 2^p$

- J. W. Cooley and J. W. Tukey (1965), Math. Computation **19**, pp. 297–301.

Algorithme: FFT

Entrée(s): X
Retour(s): Y

```
1 z ← exp(−2 i π / length(X))
2 Lz ← liste des z^k pour k variant de 0 à length(X)/2 − 1
3 return FFTrec(X, Lz)
```

Algorithme: FFTrec (récursif)

Entrée(s): X,Lz
Retour(s): Y

```
1  Xp ← termes de rangs pairs de X
2  Xi ← termes de rangs impairs de X
3  if length(Xp)>1 then
4      Lzp ← termes de rangs pairs de Lz
5      Yp ← FFTrec(Xp, Lzp)
6      Yi ← FFTrec(Xi, Lzp)
7      Yi ← produit terme-à-terme de Lz et de Yi
8      Yt ← Yp + Yi
9      Yb ← Yp − Yi
10     return join(Yt, Yb)
11 else
12     return join(Xp + Xi, Xp − Xi)
```

Calcul direct

Coût en n^2

FFT

- Coût unitaire $C_1 = 2$
- $C_p = 2\,C_{p-1} + 3 * 2^{p-1}$
- $C_p = \dfrac{3\,p-1}{2}\,2^p$
- $C_p = \dfrac{3\,\log_2(n)-1}{2}\,n$

Coût en $n\ \ln(n)$

Conteneur homogène/hétérogène

- Conteneur **homogène** : tous les éléments qu'il contient sont de même type.
  Exemple : `std::vector`, `std::map` en C++
- Conteneur **hétérogène** : ses éléments peuvent être de types différents.
  Exemple : une liste *Mathematica*®, `'list'`, `'dict'` en *Python*™

Accès direct/séquentiel

Le type d'accès est choisi en fonction du type d'opérations que l'on veut faire sur un conteneur

- **Accès direct** : par **index** ou par **clef** (**dictionnaire** ou **conteneur associatif**)

`'dict'` en *Python*™

```
dico = {} ♯ <type 'dict'>
dico['femmes'] = 256
dico['enfants'] = 312
dico['hommes']= 199
effectif = 0
for elt in dico.values() : ♯ itérateur implicite
    effectif += elt
♯ effectif vaut : 767
```

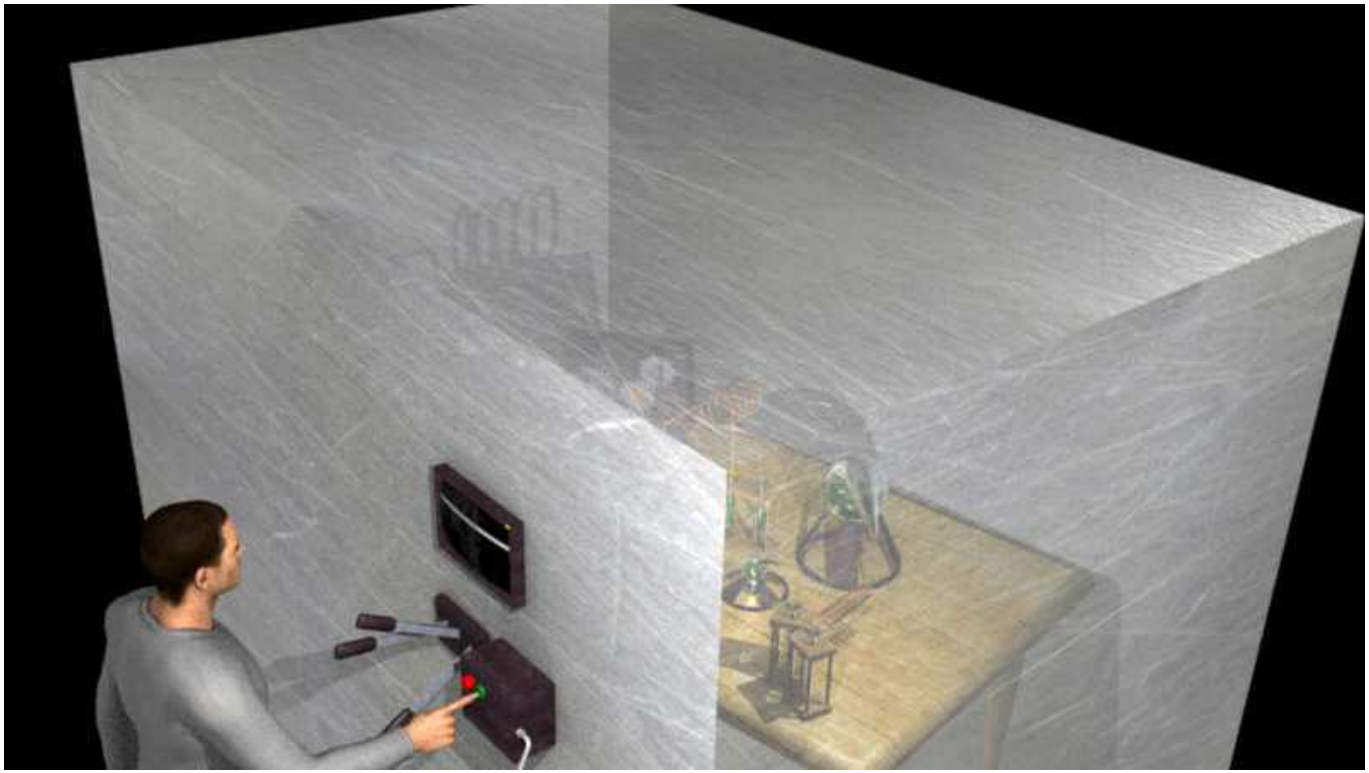
- **Accès séquentiel**
  liste chaînée simple (suivant), ou double (suivant+précédent)

Classe/objet

- Une **classe** est un modèle représentant une famille d'**objets** (type abstrait)
- Un **objet** est l'**instanciation** d'une **classe**

Attributs et méthodes

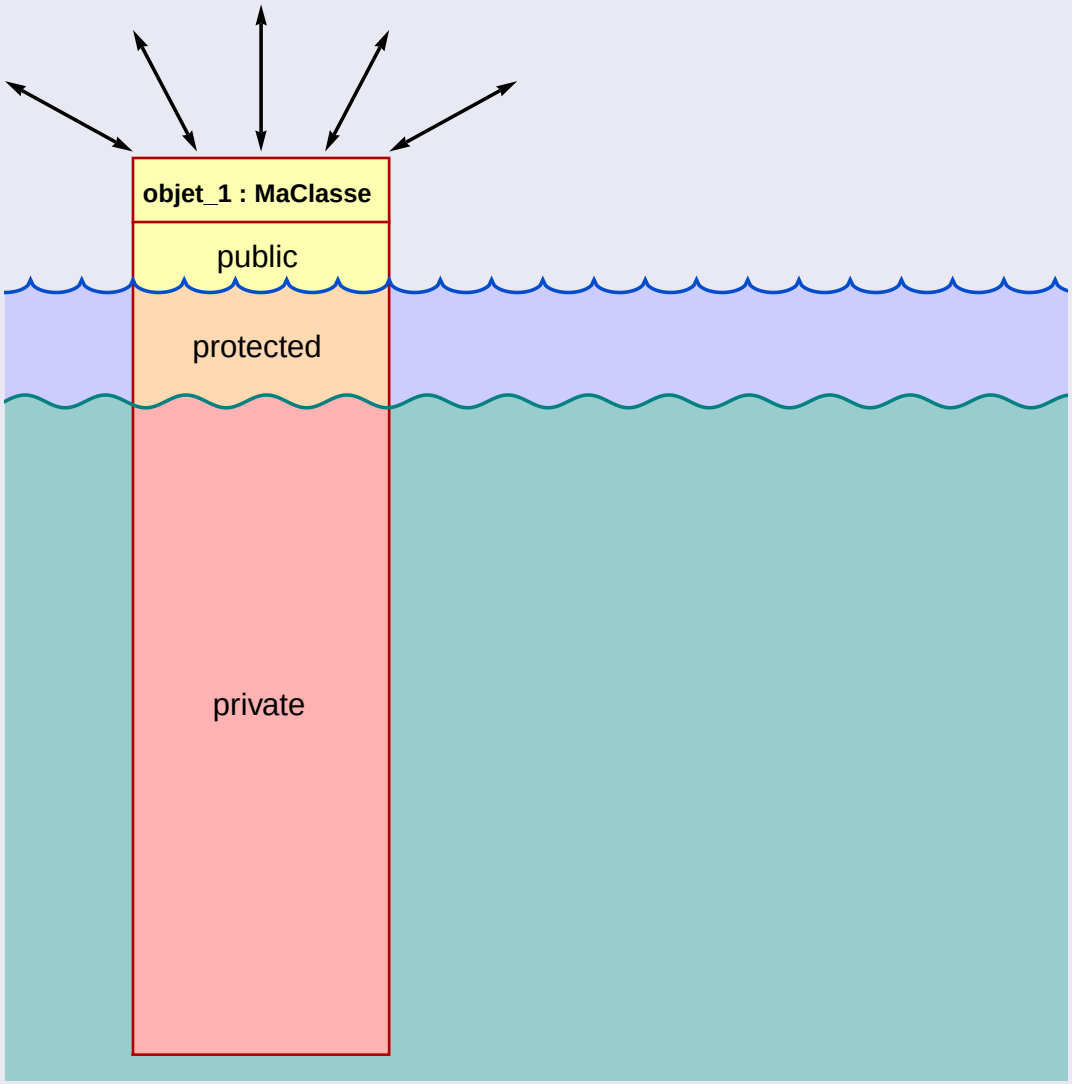
- Les **attributs** représentent les **données** contenues dans un objet
- Les **méthodes** représentent les fonctions qui s'appliquent sur l'objet



©**source** : http://fr.openclassrooms.com/informatique/cours/programmez-avec-le-langage-c/des-objets-pour-quoi-faire

Droits d'accès et encapsulation

Chaque objet **encapsule** des données



Notation UML⋆

⋆ Unified Modeling Language

+ public

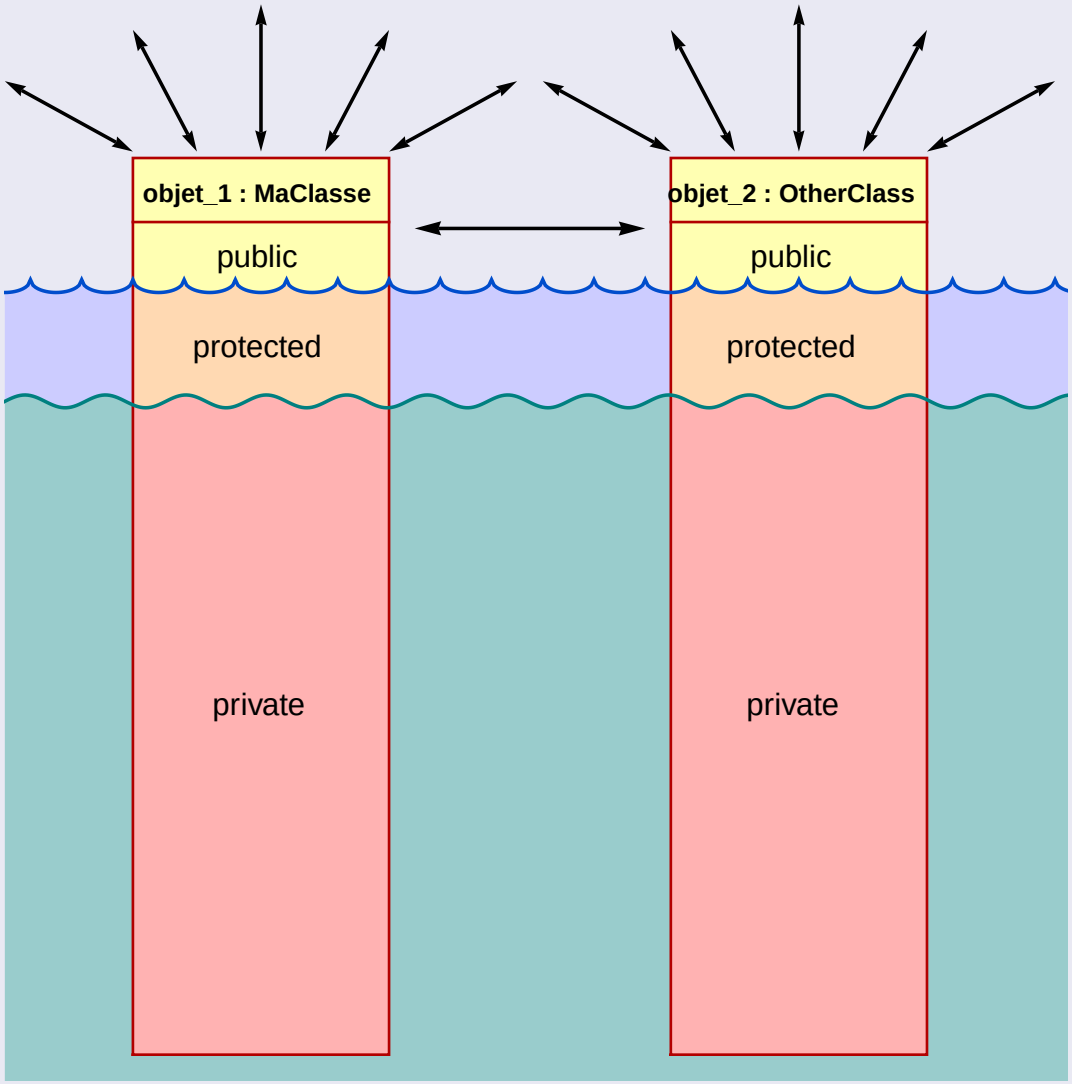
 # protected

– private

Nom de la classe : **Point**
Attributs : – x : double – y : double – label : string – red : int – green : int – blue : int – size : float
Méthodes : + tracer(out canevas : Canvas) : void + print() : void + dist(in autrePt : Point) : double + creer(in coord : double[2]) : void + setColor(in color : RGBColor) : void . . .

Droits d'accès et encapsulation

Chaque objet **encapsule** des données



Notation UML⋆

⋆ Unified Modeling Language

+ public

 # protected

– private

Nom de la classe : **Point**
Attributs : – x : double – y : double – label : string – red : int – green : int – blue : int – size : float
Méthodes : + tracer(out canevas : Canvas) : void + print() : void + dist(in autrePt : Point) : double + creer(in coord : double[2]) : void + setColor(in color : RGBColor) : void . . .

Constructeur(s)

Sert à créer l'objet.
Plusieurs méthodes peuvent exister.

Constructeur(s)

Sert à créer l'objet. Plusieurs méthodes peuvent exister.

Destructeur

Sert à libérer proprement la mémoire lors de la destruction de l'objet.

Constructeur(s)

Sert à créer l'objet. Plusieurs méthodes peuvent exister.

Destructeur

Sert à libérer proprement la mémoire lors de la destruction de l'objet.

Getter(s)

Sert à lire des attributs dans la zone privée.

Constructeur(s)

Sert à créer l'objet. Plusieurs méthodes peuvent exister.

Destructeur

Sert à libérer proprement la mémoire lors de la destruction de l'objet.

Getter(s)

Sert à lire des attributs dans la zone privée.

Setter(s)

Sert à modifier les attributs.

Attention :

- Cela donne accès à la zone privée.
- Il arrive souvent que les attributs soient interdépendants (redondance pour gagner en calcul).

Constructeur(s)

Sert à créer l'objet. Plusieurs méthodes peuvent exister.

Destructeur

Sert à libérer proprement la mémoire lors de la destruction de l'objet.

Getter(s)

Sert à lire des attributs dans la zone privée.

Setter(s)

Sert à modifier les attributs.

Attention :

- Cela donne accès à la zone privée.
- Il arrive souvent que les attributs soient interdépendants (redondance pour gagner en calcul).

Exemple en *Python*™

```
class Point :
    # Constructeur :
    def __init__(self, xx=0, yy=0, ee='O', ss=1, cc='black') :
        self.__x = xx # abscisse
        self.__y = yy # ordonnée
        self.__label = ee # nom affiché
        self.__size = ss # taille
        self.__color = cc # couleur
    # Getters :
    def x(self) : return self.__x
    def y(self) : return self.__y
    def coord(self) : return [self.__x, self.__y]
    def label(self) : return self.__label
    ...
    # Setters :
    def setx(self,x) : self.__x = x
    def sety(self,y) : self.__y = y
    ...
    # Autres méthodes :
    def prt_prop(self) :
        print "Point", self.label()," : abscisse ",\
            self.x(), " ; ordonnée ", self.y(),
```

```
# Main :
pt1 = Point(50, 50, 'A', 8, 'red')
pt1.prt_prop()
pt2 = Point(200, 150, 'B', 5, 'maroon')
pt2.prt_prop()
...
```

Surcharge d'opérateur

- arithmétiques (+, −, *, /,. . .)
- de comparaison (==, >, <, . . .)
- d'affectation (=)
- de flux (<<, >>)

Exemple en *Python*™

```
class Vecteur :
    ♯ Constructeur :
    def __init__(self, vx=0, vy=0) :
        self.__x = vx ♯ abscisse
        self.__y = vy ♯ ordonnée
    . . .
    ♯ Surcharge d'opérateurs :
    def __eq__(self, v) : return (self.__x==v.__x) and (self.__y==v.__y) ♯ teste l'égalité de 2 vecteurs
    def __add__(self, v) : return Vecteur(self.__x+v.__x, self.__y+v.__y) ♯ retourne le vecteur somme
    def __sub__(self, v) : return Vecteur(self.__x−v.__x, self.__y−v.__y) ♯ retourne le vecteur différence
```

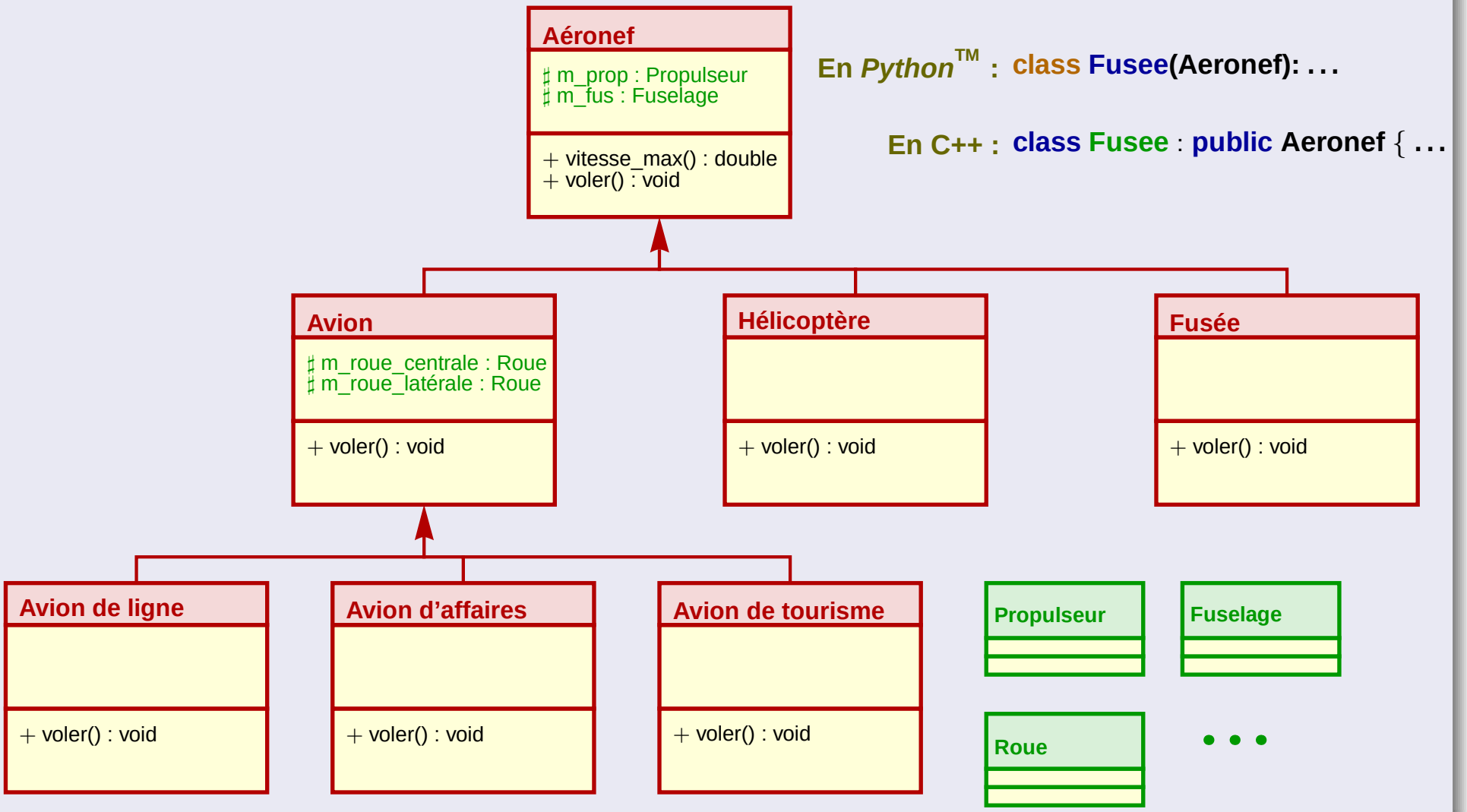
« obj1 + obj2 » est équivalent à « obj1.__add__(obj2) »
« obj1 − obj2 » est équivalent à « obj1.__sub__(obj2) »
« obj1 * obj2 » est équivalent à « obj1.__mul__(obj2) »
« obj1 / obj2 » est équivalent à « obj1.__div__(obj2) »
« obj1 == obj2 » est équivalent à « obj1.__eq__(obj2) »
. . .

```
class Point :
    . . .
    ♯ Surcharge d'opérateurs :
    def __add__(self, obj) :
        if isinstance(obj, Vecteur) : return Point(self.__x+obj.x(), self.__y+obj.y())
        elif isinstance(obj, Point) : return Point(0.5*(self.__x+obj.__x), 0.5*(self.__y+obj.__y))
        else : return None
    def __sub__(self, obj) :
        if isinstance(obj, Vecteur) : return Point(self.__x−obj.x(), self.__y−obj.y())
        elif isinstance(obj, Point) : return Vecteur(self.__x−obj.__x, self.__y−obj.__y)
        else : return None
```

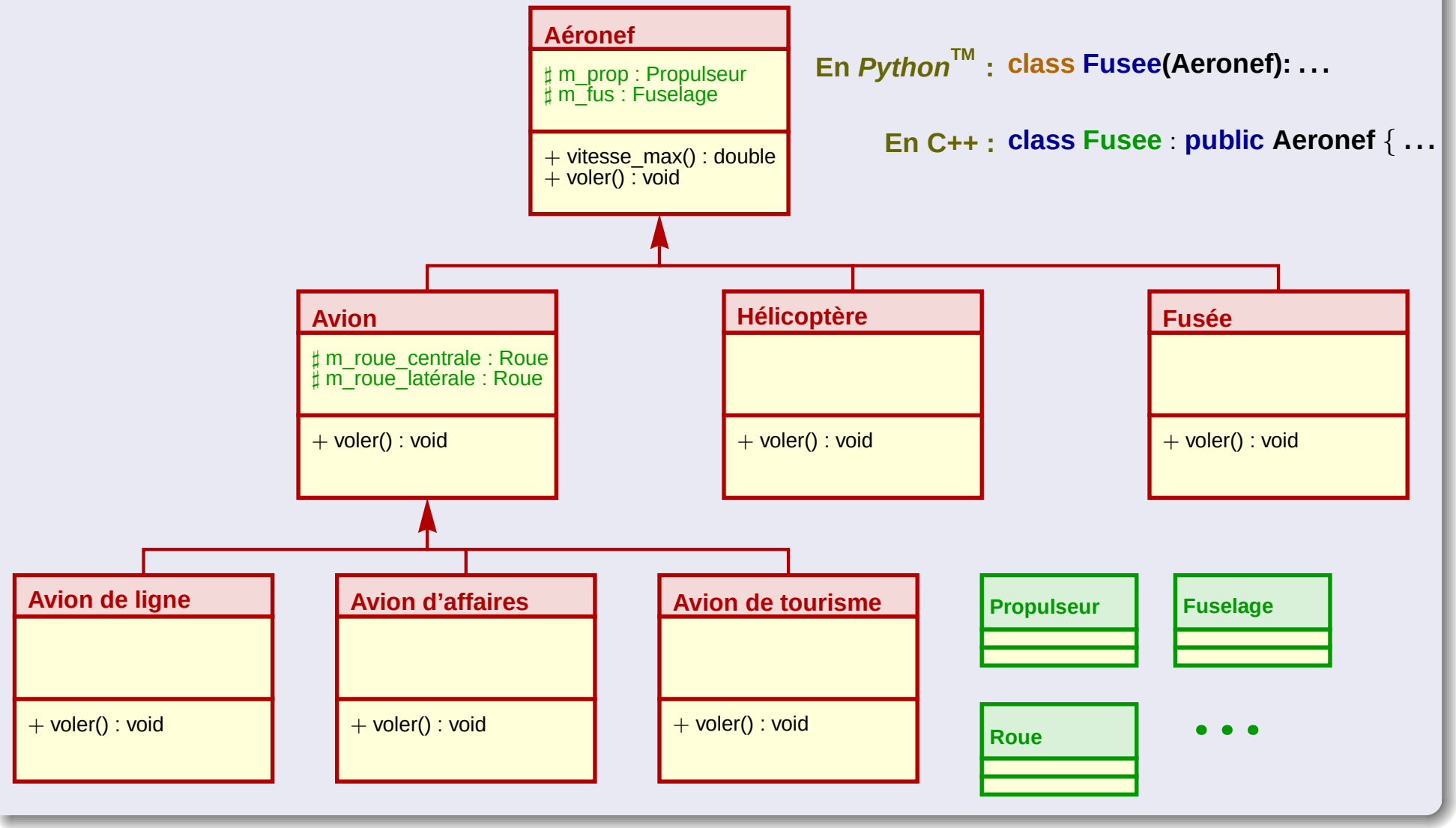
Héritage

- L'**héritage** est **LE** mécanisme de transmission des propriétés (attributs et méthodes) d'une classe à une autre.
- La **classe de base** (classe mère, parente) transmet toutes ses propriétés transmissibles (public, protected) aux classes dérivées (classe fille, enfant).
- La **classe dérivée** :
  - ✓ **ne peut pas accéder aux membres privés (private) de la classe de base** ;
  - ✓ possède ses propres attributs et méthodes, que la classe de base ne connaît pas ;
  - ✓ peut redéfinir (améliorer, spécialiser, . . .) les **méthodes héritées** de la classe de base.
- Tout ce que la classe de base sait faire, la classe dérivée sait le faire ; éventuellement elle sait le faire « mieux » ou « différemment ».

Héritage (diagramme UML)



Héritage (diagramme UML)



« **Modélisation Orientée Objet** » : *cf.* module de deuxième année.