

# Fondamentaux du langage Python

- Variables / Types
- Entrées / Sorties
- Structures de contrôle
- Fonctions

Radouan Dahbi  
[r.dahbi@caplogy.com](mailto:r.dahbi@caplogy.com)



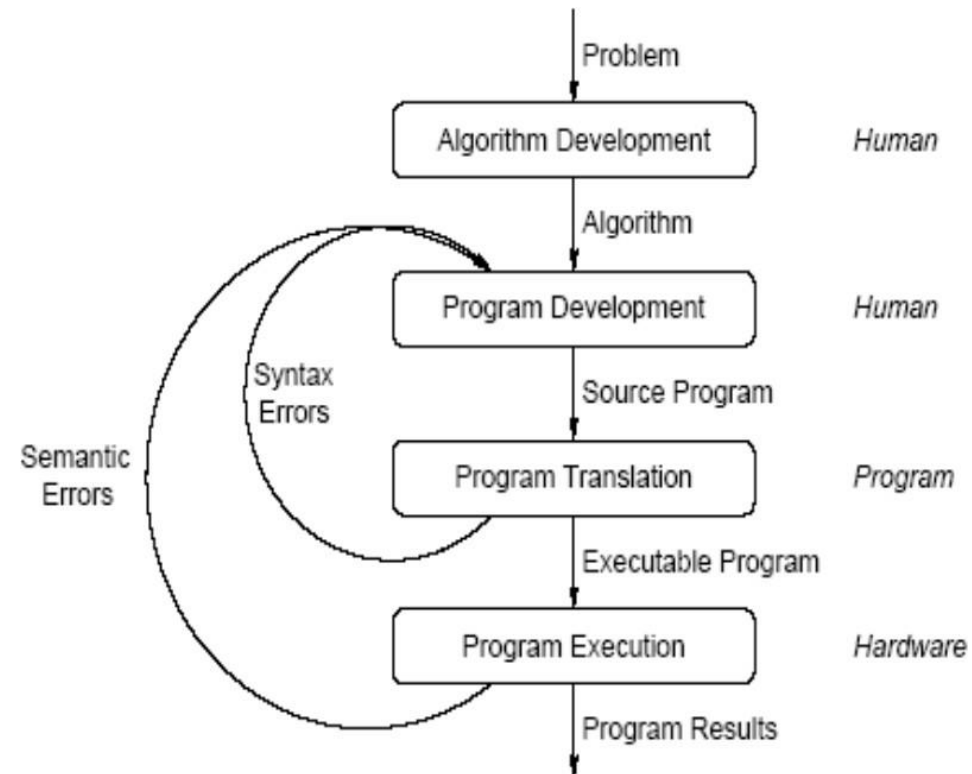
01

# Le langage Python



# La programmation

## Ce qu'est programmer



# La programmation

## Erreurs syntaxiques Vs Erreurs sémantiques

- **Erreur de Syntaxe** : violation d'une règle du langage de programmation (grammaire).
  - Exemple: "Moi parle Anglais bien."
  - Utiliser des mots clef valides mais au mauvais endroit
  - Détecté par un compilateur
- **Erreur de Sémantique** : erreur logique :
  - Exemple: "Cette phrase est écrite en Italien"
  - Un programme syntaxiquement correct mais ne fournissant pas le résultat attendu
  - L'utilisateur observe la sortie de l'exécution d'un programme

# Le langage Python

## Définition

- Développé par Guido van Rossum au début des années 1990.
- A télécharger <http://www.python.org>
- Langage interprété : exécution du code source ligne par ligne, en utilisant un interpréteur
- Langage orienté objet : chaque élément est considéré comme objet, c-à-d une entité qui combine des variables et des fonctions
- Typage dynamique : le type d'une variable est déterminé automatiquement en fonction de la valeur qui lui est attribuée

# Le langage Python

## Modes de programmation en python

Dans ce cours, seul le mode IDE sera abordé.

L'IDE à utiliser est: PyCharm





**02**

# **Les bases de la programmation Python**



# Les bases de la programmation Python

## Les types primitifs

Primitive	Signification	Taille (en octets)	Plage de valeurs acceptée
int	Entier	4	-2 147 483 648 à 2 147 483 647
float	flottant (réel)	8	$10^{-308}$ à $10^{308}$
str	Chaîne de caractères		
bool	booléen	1	False ou True (toute valeur différente de 0 est considérée comme True)

Syntaxe exacte du type

Informations sur le type



# Les bases de la programmation Python

## Manipulation des variables

### Déclaration d'une variable

Typage dynamique : pas de déclaration explicite de la variable.

Python **devine** le type de la variable « Nom\_variable » à partir du type de la valeur affectée.

**Nom\_variable** = **valeur**

Nom symbolique donnée à une variable: choisir des noms significatifs.

Initialisation de la variable: première valeur attribuée à une variable.

# Les bases de la programmation Python

## Manipulation des variables

### Exemples

```
num_etudiant = 5 # num_etudiant sera considérée de type int  
PI = 3.14 # PI sera considérée comme une variable de type float  
prenom = "Maria" # prenom est une variable de str ici  
est_valide = True # Cette variable est de type bool
```

# Les bases de la programmation Python

## Nommage des variables

### Il est autorisé de:

- Utiliser les lettres minuscules (a – z)
- Utiliser les lettres majuscules (A – Z)
- Utiliser les chiffres (0 – 9)
- Le caractère de soulignement (\_)

### Il n'est pas autorisé de:

- Commencer le nom d'une variable par un chiffre
- Utiliser des espaces dans les noms des variables
- Utiliser des accents
- Utiliser les mots réservés du langage: print, def, and, or, for, ...

Python est sensible à la casse.

### Exemple:

Les variables nommées NOTE, Note, note sont différentes.

# Les bases de la programmation Python

## Les opérateurs

### L'affectation


Symbolisée par le caractère `=`. Elle permet l'attribution d'une valeur à une variable.

Exemple: `x = 5`

### L'affectation multiple

Python donne la possibilité d'affecter des valeurs à plusieurs variables sur une seule ligne.

Exemple:

`x, y, z = 5, 3.5, "test"`  `x = 5`  
Equivalent à `y = 3.5`  
`z = "test"`

`a = b = 3`  `b = 3`  
Equivalent à `a = b`

# Les bases de la programmation Python

## Les opérateurs

### Les opérateurs arithmétiques

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	$x+3$	10
-	opérateur de soustraction	Soustrait deux valeurs	$x-3$	4
*	opérateur de multiplication	Multiplie deux valeurs	$x*3$	21
/	opérateur de division	Divise deux valeurs	$x/3$	2.3333333
%	Opérateur du modulo	Reste de la division	$x\%3$	1
//	Division entière	Quotient d'une division	$x//3$	2
**	Opérateur de puissance	Calcul de la puissance	$x**2$	49

# Les bases de la programmation Python

## Les opérateurs

### Les opérateurs arithmétiques composés

Opérateur	Opération composée	Opération normale
<code>+=</code>	<code>X += Y</code>	<code>X = X + Y</code>
<code>-=</code>	<code>X -= Y</code>	<code>X = X - Y</code>
<code>*=</code>	<code>X *= Y</code>	<code>X = X * Y</code>
<code>/=</code>	<code>X /= Y</code>	<code>X = X / Y</code>
<code>//=</code>	<code>X //= Y</code>	<code>X = X // Y</code>
<code>%=</code>	<code>X %= Y</code>	<code>X = X % Y</code>
<code>**=</code>	<code>X **= Y</code>	<code>X = X ** Y</code>

# Les bases de la programmation Python

## Les opérateurs

### Opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat
<code>==</code> <b>A ne pas confondre avec le signe d'affectation (=)!!</b>	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	<code>x==3</code>	Retourne True si x est égal à 3, sinon False
<code>&lt;</code>	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	<code>x&lt;3</code>	Retourne True si x est inférieur à 3, sinon False
<code>&lt;=</code>	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	<code>x&lt;=3</code>	Retourne True si x est inférieur ou égal à 3, sinon False
<code>&gt;</code>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	<code>x&gt;3</code>	Retourne True si x est supérieur à 3, sinon False
<code>&gt;=</code>	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	<code>x&gt;=3</code>	Retourne True si x est supérieur ou égal à 3, sinon False
<code>!=</code>	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	<code>x!=3</code>	Retourne True si x est différent de 3, sinon False

# Les bases de la programmation Python

## Les opérateurs

### Opérateurs logiques

Opérateur	Dénomination	Effet	Syntaxe
or	OU logique	Vérifie qu'il y a au moins une condition réalisable	((condition1) or (condition2))
and	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1) and (condition2))
not	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur True si la variable vaut False, False si elle vaut True)	(not condition)



# Les bases de la programmation Python

## Les opérateurs

### Opérateurs d'appartenance

Opérateur	Dénomination	Effet	Syntaxe
in	Test d'appartenance	True si la valeur / variable se trouve dans la séquence	5 in x
not in	Test de non appartenance	True si la valeur / variable ne se trouve pas dans la séquence	5 not in x

# Les bases de la programmation Python

## Les opérateurs

### Opérateurs d'identité

Opérateur	Dénomination	Effet	Syntaxe
is	Test d'égalité de types non primitifs	True si les opérandes sont identiques (se référer au même Type)	x is True
is not	Test de non égalité de types non primitifs	True si les opérandes ne sont pas identiques (pas de référence pour le même Type)	x is not True

# Les bases de la programmation Python

## La concaténation

- Mettre bout à bout deux chaînes de caractères
- L'opérateur **+** reliant deux chaînes de caractères fait effet de concaténation

Expression arithmétique    **+**    Expression arithmétique  
Opérateur d'addition

### Exemple

```
a = 3  
b = a + 3
```

b = 6

Chaîne de caractères    **+**    Chaîne de caractères  
Opérateur de concaténation

### Exemple

```
mot1 = "cours"  
mot2 = "python"  
mot = mot1 + mot2
```

mot = « courspython »

# Les bases de la programmation Python

## La répétition

- L'opérateur `*` appliqué à une chaînes de caractères permet de répéter cette chaîne de caractères un nombre de fois

Expression  
arithmétique



Expression  
arithmétique

Opérateur de  
multiplication

Exemple

```
a = 3  
b = a * 3
```

b = 9

Nombre



Chaîne de  
caractères

Opérateur de  
répétition

Exemple

```
mot1 = "cours"  
mot = 3 * mot1
```

mot = « courscourscours »

# Les bases de la programmation Python

## Casting

- Forcer la conversion d'une variable d'un type donné vers un autre type.
- Réalisable en utilisant les méthodes:
  - `int()`
  - `float()`
  - `str()`

### Exemples

```
y = int(2.8) # y sera égal à 2
z = int("3") # z sera égal à 3
a = float("3") # a sera égal à 3.0
w = float("4.2") # w sera égal à 4.2
b = str(3.0) # b sera égal à '3.0'
```



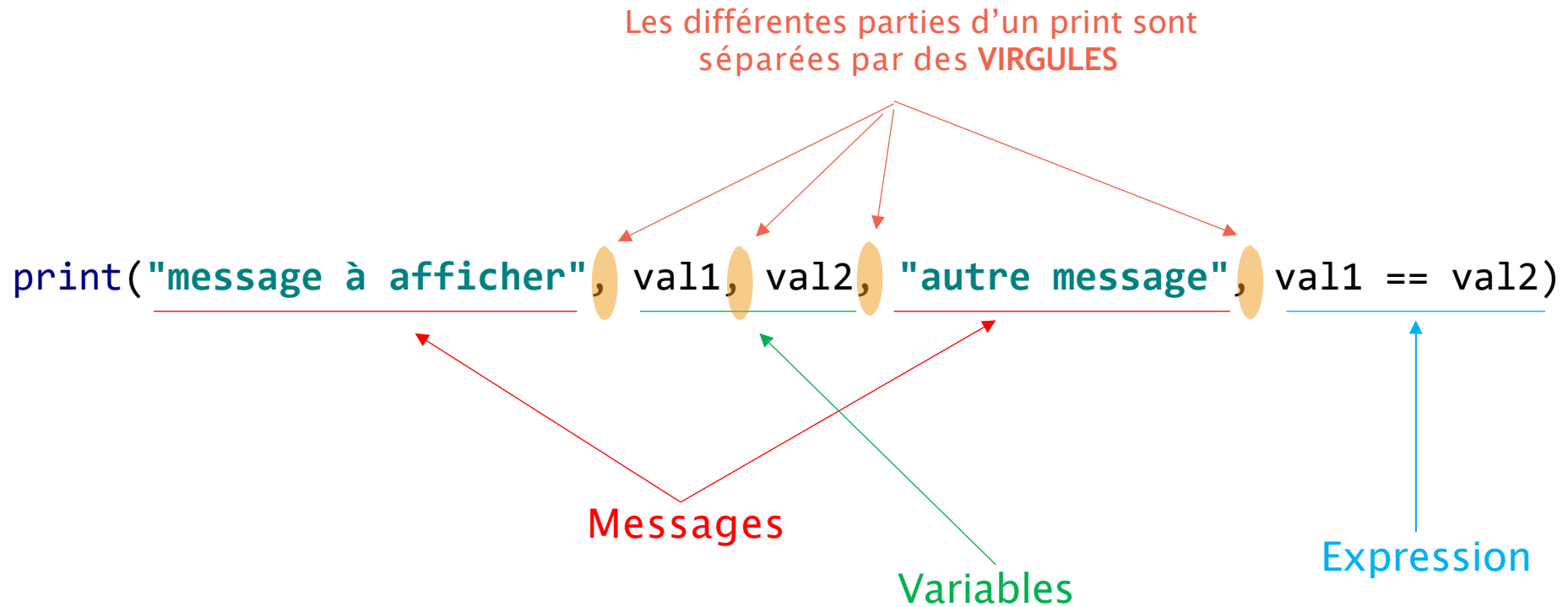


**03**

## **Les entrées / sorties en langage Python**

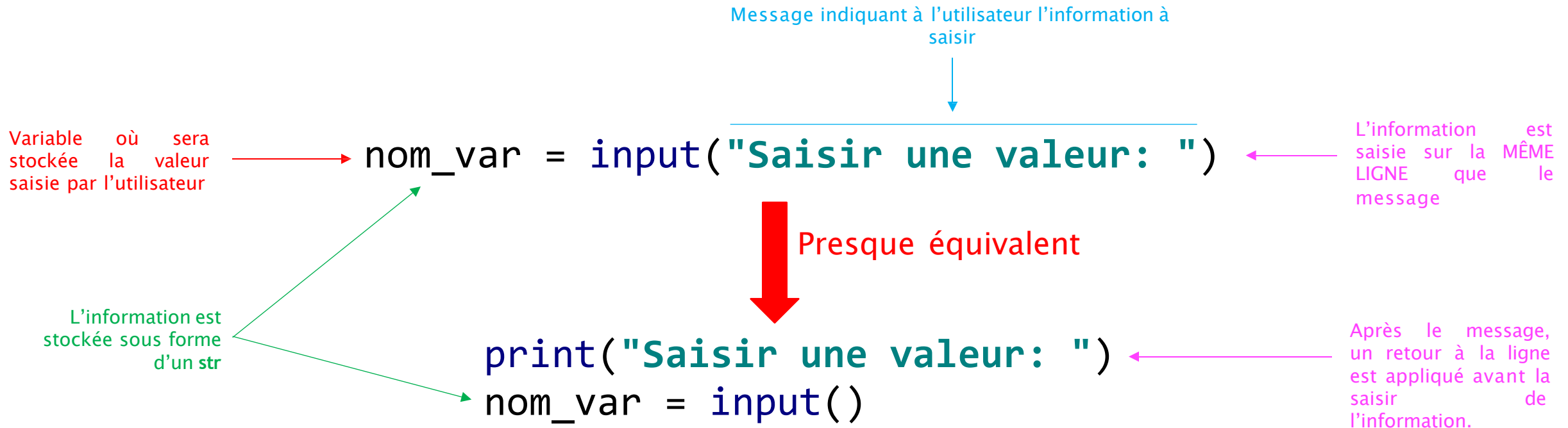
# Les entrées / sorties en Python

## Affichage à l'écran



# Les entrées / sorties en Python

## Saisie de données





# Les entrées / sorties en Python

## Saisie de données

Comment faire pour saisir des données numériques ?

```
nom_var_int = int(input("Saisir une valeur: "))
```

↑  
Variable de type int

```
nom_var_float = float(input("Saisir une valeur: "))
```

↑  
Variable de type float



**04**

## **Les structures de contrôle**



# Les structures de contrôle

## Les structures conditionnelles

### ■ if : syntaxe

Mot clé : OBLIGATOIRE

**if** condition  
action(s)

Indentation: OBLIGATOIRE. Toutes les actions à faire si la condition du if est True doivent être dans un bloc indenté

Peut être:

- Condition simple: expression de comparaison
- Condition composée: expression logique

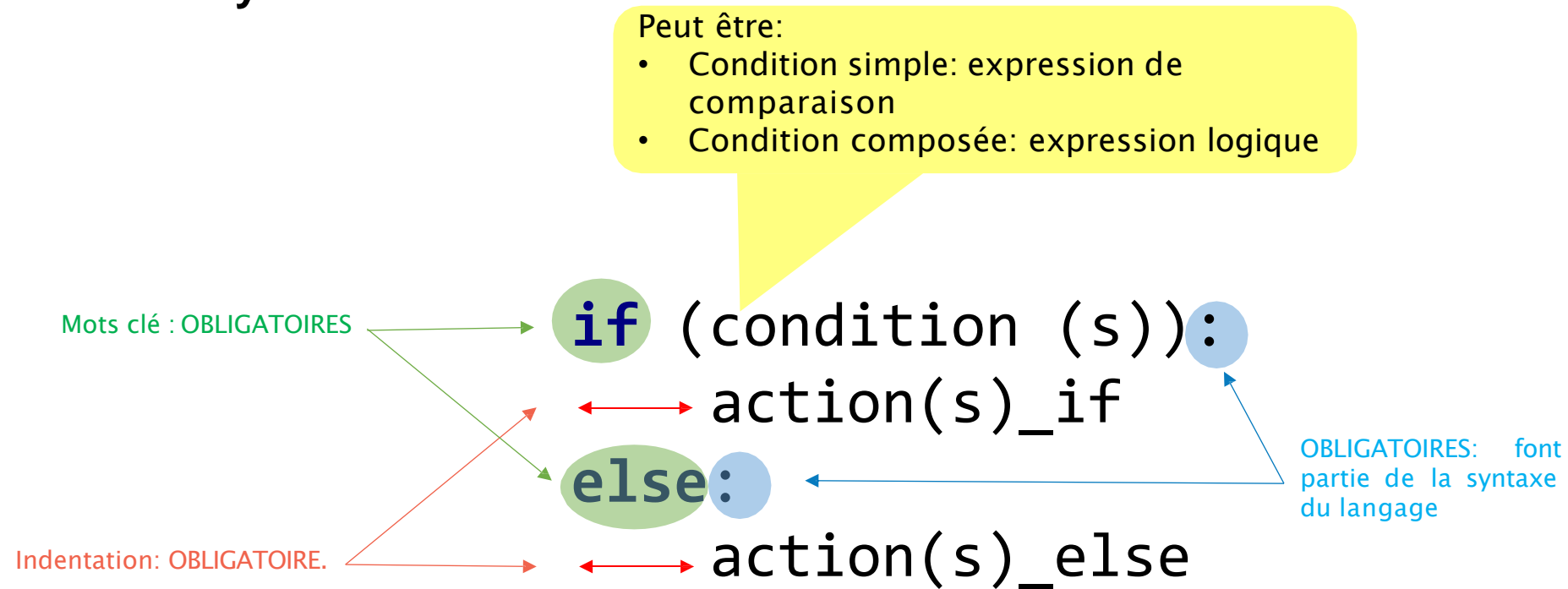
OBLIGATOIRE: fait partie de la syntaxe du langage

Une ou plusieurs action(s) exécutée(s) uniquement si condition est True

# Les structures de contrôle

## Les structures conditionnelles

### ■ if - else: syntaxe



# Les structures de contrôle

## Les structures conditionnelles

### ■ if – elif - else: syntaxe

Mots clé : OBLIGATOIRES

```
if condition(s)1 :  
    action(s)_if  
elif condition(s)2 :  
    action(s)_elif  
...  
elif condition(s)N :  
    action(s)_elif :  
else :  
    action(s)_else
```

Syntaxe utilisée lorsqu'il y a plusieurs cas à tester: cas de « if » imbriqués

Indentation: OBLIGATOIRE.

OBLIGATOIRES: font partie de la syntaxe du langage

# Les structures de contrôle

## Les structures conditionnelles

### ■ if – elif - else: exemples

```
a = int(input("Saisir une valeur: "))
if (a > 0):
    print(a, "est strictement positif")
else:
    if (a < 0):
        print(a, "est strictement négatif")
    else:
        print(a, "est nul")
```

```
a = int(input("Saisir une valeur: "))
if (a > 0):
    print(a, "est strictement positif")
elif (a < 0):
    print(a, "est strictement négatif")
else:
    print(a, "est nul")
```

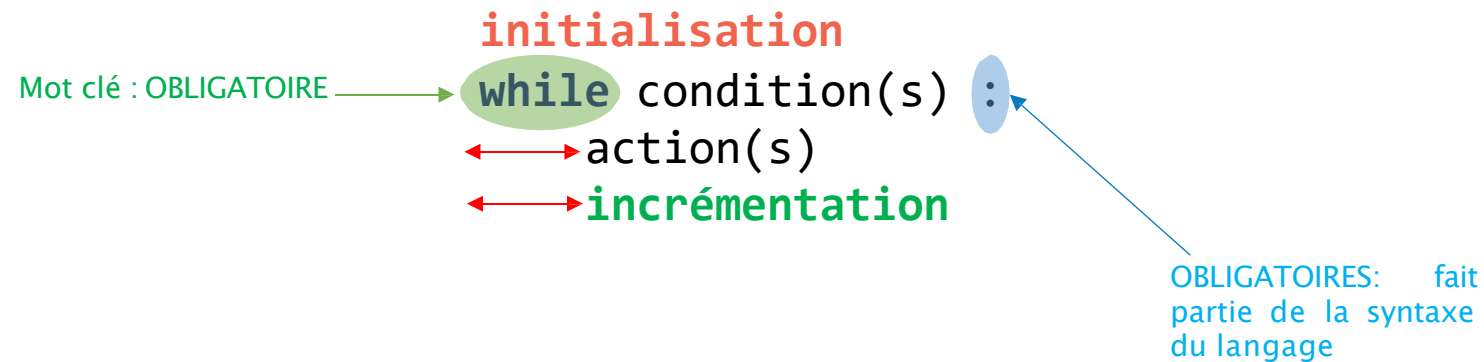


Peut également s'écrire comme ça

# Les structures de contrôle

## Boucles

### ■ while: syntaxe



- ✓ Exécuter le corps de la boucle (`action(s)`) tant que la condition est vérifiée (le test vaut **True**)
- ✓ La condition est évaluée **avant** d'exécuter le corps de la boucle
- ✓ Le corps de la boucle peut contenir une ou plusieurs autres boucles: **Boucles imbriquées**

# Les structures de contrôle

## Boucles

### ■ while: exemples

```
x = 2
while x < 5 :
    print(x, " est dans la boucle")
    x = x + 1
```

Exemple d'utilisation d'une  
boucle while

```
x = 2
while x < 5 :
    y = 1
    while y < 5 :
        print(x + y)
        y = y + 1
    x = x + 1
```

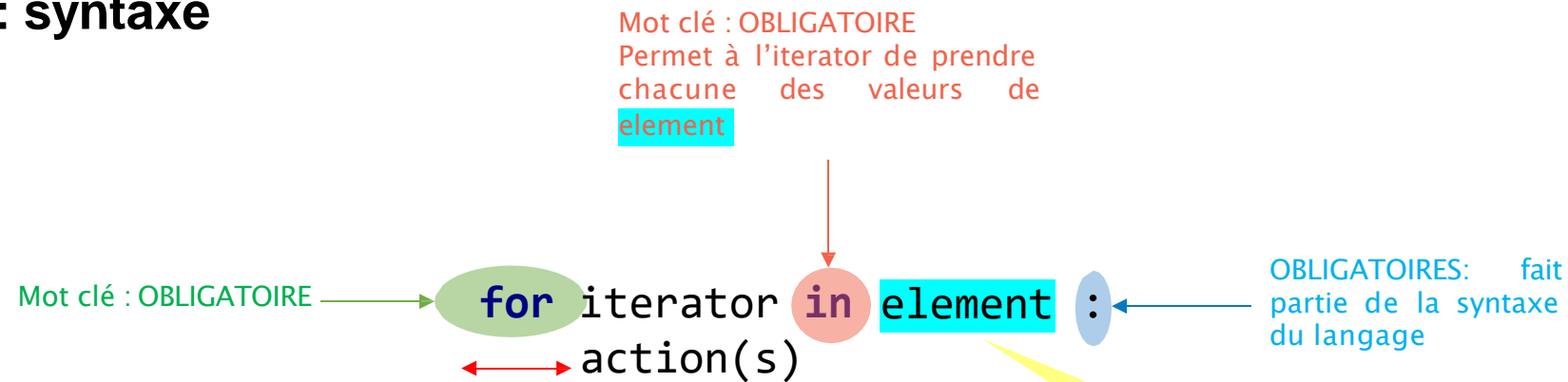
Exemple d'utilisation de  
boucles while imbriquées



# Les structures de contrôle

## Boucles

### ■ for: syntaxe



« **element** » peut être :

- Un intervalle de valeurs
- Une chaîne de caractères
- Un tableau (liste en python)

# Les structures de contrôle

## Boucles

### ■ for: La fonction range()

```
for iterator in range(nombre) :  
    action(s)
```

- iterator prend par défaut les valeurs de 0 à **nombre** - 1(inclus)
- iterator s'incrmente de 1 à chaque tour de boucle.

```
for iterator in range(borne_inf, borne_sup) :  
    action(s)
```

- iterator prend par défaut les valeurs de borne\_inf à **borne\_sup** - 1(inclus)
- iterator s'incrmente de 1 à chaque tour de boucle.

```
for iterator in range(borne_inf, borne_sup, pas) :  
    action(s)
```

- iterator prend par défaut les valeurs de borne\_inf à **borne\_sup** - 1(inclus)
- iterator s'incrmente de **pas** à chaque tour de boucle.

# Les structures de contrôle

## Boucles

### ■ for: exemples

range(nombre)	range(borne_inf, borne_sup)	range(borne_inf, borne_sup, pas)	str
<code>for i in range(4):   print(i)</code>	<code>for i in range(2, 7):   print(i)</code>	<code>for i in range(2, 10, 2):   print(i)</code>	<code>for i in "bonjour":   print(i)</code>
0 1 2 3	2 3 4 5 6	2 4 6 8	b o n j o u r

# Les fonctions

## Rappel

- Une fonction est un sous programme permettant de réaliser une tâche bien définie.
- **Fonctionnement :**
  - Avant d'utiliser une fonction , il faut d'abord **la définir** ;
  - Au moment de l'utilisation, il faut effectuer **un appel** de la fonction

# Les fonctions

## Définition d'une fonction sans sortie : Syntaxe

```
def nom_fonction ( param1, param2, ... )
```



↔ Corps de la fonction

## exemple

```
# Fonction sans sortie à 2 paramètres
def add_numbers (a, b):
    res = a + b
    print("{} + {} = {}".format(a, b, res))
```

# Les fonctions

## Appel d'une fonction sans sortie : Syntaxe

Il est possible d'appeler une fonction n'importe où **après** avoir quitté la fonction



**MAIS** Il n'est pas possible d'appeler une fonction **avant** sa définition



**nom\_fonction** ( **arg1, arg2, ...** )

- Les argument doivent être du même type que les paramètres
- L'ordre sémantique des arguments doit respecter l'ordre des paramètres.

## exemple

```
# Fonction sans sortie à 2 paramètres
def add_numbers (a, b):
    res = a + b
    print("{} + {} = {}".format(a, b, res))

# Appel d'une fonction sans sortie avec 2 arguments
add_numbers(1, 3)
```

1 + 3 = 4

# Les fonctions

## Définition d'une fonction avec sortie : Syntaxe

```
def nom_fonction ( param1, param2, ... ) :
```

↔ Corps de la fonction

↔ **return** résultat

## exemple

```
# Fonction avec sortie à 2 paramètres
def add_numbers (a, b):
    res = a + b
    return res
```

# Les fonctions

## Appel d'une fonction sans sortie : Syntaxe

res = nom\_fonction ( arg1, arg2, ... )

Le résultat retourné par la fonction doit être accueilli dans une variable à l'appel

### exemple

```
# Fonction avec sortie à 2 paramètres
def add_numbers (a, b):
    res = a + b
    return res

# Appel d'une fonction avec sortie à 2 arguments
x = int(input("Saisir un entier : "))
y = int(input("Saisir un autre entier : "))
val = add_numbers(x, y)
print("{} + {} = {}".format(x, y, val))
```

```
Saisir un entier : 1
Saisir un autre entier : 3
1 + 3 = 4
```



# Les fonctions

## Les sorties multiples:

- Généralement, une fonction ne peut retourner qu'une seule sortie.
- En Python, il est possible de retourner plusieurs sortie **Et c'est UNIQUEMENT en Python !!!**



## Exemple

```
def min_max(a, b):  
    minimum, maximum = a, b  
    if b < a :  
        minimum, maximum = b, a  
    return minimum, maximum
```

```
Saisir un entier : 5  
Saisir un autre entier : 7  
La valeur minimum est 5 et la valeur maximum est 7
```

```
# Appel de la fonction min_max  
x = int(input("Saisir un entier : "))  
y = int(input("Saisir un autre entier : "))  
mini, maxi = min_max(x, y)  
print("La valeur minimum est {} et la valeur maximum est {}".format(mini, maxi))
```