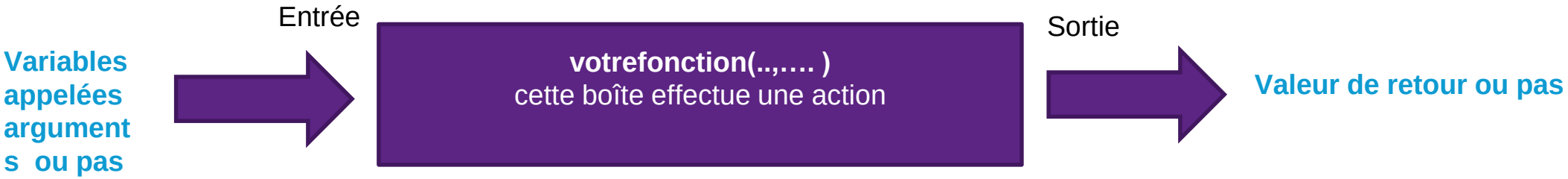
NOTE 4
Définir ses propres fonctions

ESIEE[it]
L'école de l'expertise numérique
une école de la
CCI PARIS ILE-DE-FRANCE

Principe des fonctions

Les fonctions sont utiles pour réaliser plusieurs fois la même opération au sein d'un programme

Une fonction est une sorte de boîte noire dans laquelle on peut mettre des variables en entrée ou pas. Ces variables peuvent être de n'importe quel type et cette boîte peut vous rendre une variable ou un objet, ou rien du tout



Par exemple, si vous appelez la fonction len() de la manière suivante : len([1,2,3])

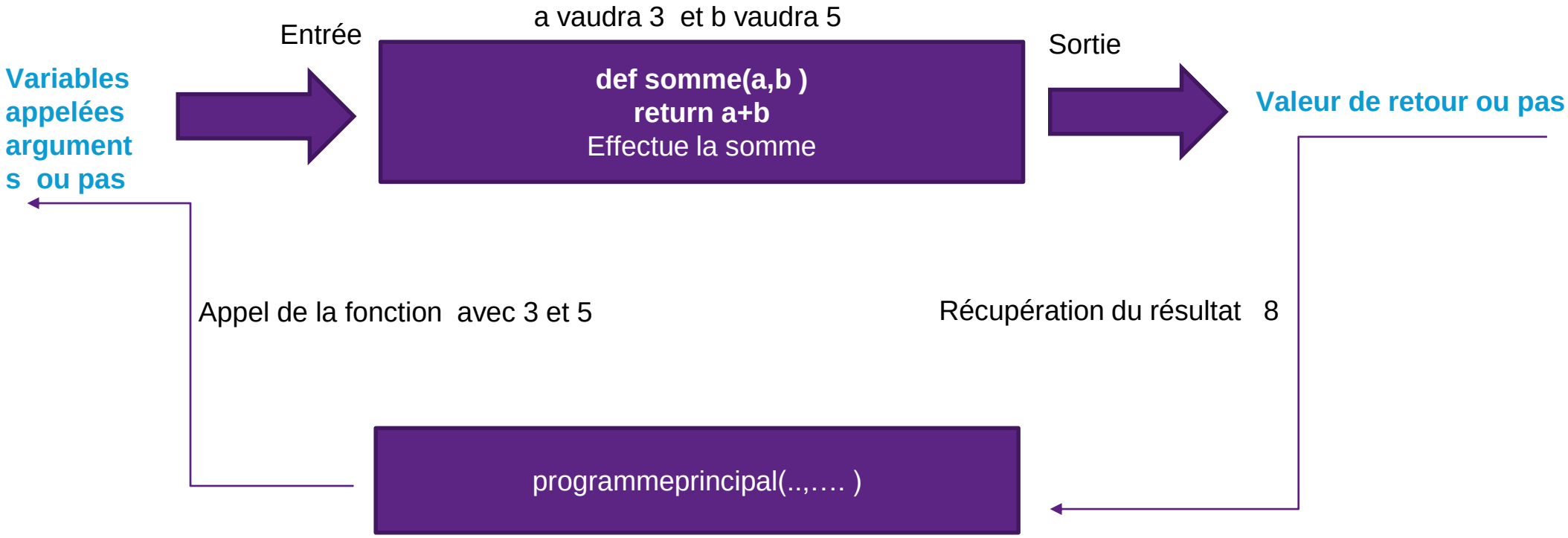
La liste [1,2,3] est le paramètre d'entrée

Action
=
Calcule la longueur de la liste passée en paramètre

3 est la valeur de retour

Principe des fonctions

Le programme principal appelé **Main** va appeler les fonctions pour leur déléguer une partie du travail



Exemple d'instructions du programme principal

```
x=3
y=5
w=somme(x,y)
print('w vaut '+w)
```

Le programme principal affichera :

w vaut 8

fonctions sans paramètre d'entrée

On utilise le mot clef **def** pour déclarer une fonction
On indente les instructions de la fonction

Syntaxe

```
def mafonction()  :
        instructions1
        instructions2
```

Exemple

```
def affichebonjour()  :
    print('bonjour')
```

Utilisation de la fonction dans le main

```
affichebonjour()
```

Résultat

```
bonjour
```

Il est possible de déclarer des variables dans une fonction,
On les appelles **variables locales** car elles n'existent pas après la sortie de la fonction
Les **variables globales** sont déclarées dans le main est connue de partout

Exemple

```
def afficheA()  :
        a=5
        a=7
        print(a)

a=3
print(a)
afficheA()
print(a)
```

résultat

```
3 #valeur du a global
7 #valeur du a local
3 #valeur du a global
```

fonctions avec paramètres d'entrée

Le nombre d'arguments que l'on peut passer à une fonction est variable.

vous n'êtes pas obligé de préciser le type des arguments que vous lui passez, du moment que les opérations sont valides.

Exemple

```
def somme(a,b)  :
      print('la somme vaut '+ (a+b))

#on lui passe 2 entiers
>>somme(3,4)
la somme vaut 7
#on lui passe 2 listes
>>somme([1,2], [3,4])
la somme vaut [1,2, 3,4]
```

- **Paramètres optionnels et valeur par défaut**

  il est aussi possible de passer un ou plusieurs argument(s) de manière facultative et de leur attribuer une valeur par défaut

Exemple

```
def somme(a,b=2)  :
      print('la somme vaut '+ (a+b))

#on lui passe 1 entier et l'autre aura la valeur par défaut
>>somme(3)
la somme vaut 5
```

Exemple

```
def somme(a,b=2,c=3)  :
      print('la somme vaut '+ (a+b+c))

#on lui passe 2 entiers et on précise lequels des 2 arguments non obligatoires et passés
>>somme(3, b=6)
la somme vaut 12
```

Paramètres d'entrée issues d'une liste ou d'un dictionnaire

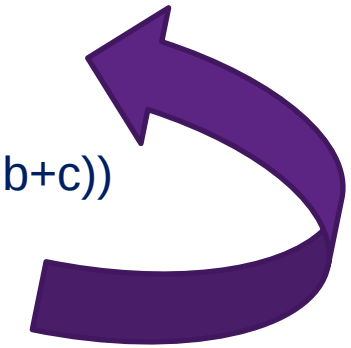
A la place d'un plusieurs arguments, on peut passer une liste qui sera dépliée en utilisant la notation * lors de l'appel de la fonction, de même avec ** dans le cas d'un dictionnaire

- **Notation * pour une liste**

Exemple

```
def somme(a,b,c) :
      print('la somme vaut '+ (a+b+c))

>>maliste=[1,2,3]
>>somme(**maliste)
la somme vaut 6
```



a=1
b=2
c=3

Au moment de l'appel de la fonction somme,
maliste est dépliée pour remplir les valeurs de a, b et c avec les valeurs de la liste

- **Notation ** pour un dictionnaire**

Exemple

```
def somme(a,b,c) :
      print('la somme vaut '+ (a+b+c))

>>mondico={'clef1':1,'clef2':2, 'clef3': 3}
>>somme(**mondico)
la somme vaut 6
```

Au moment de l'appel de la fonction somme,
Les valeurs de mondico servent à remplir les valeurs de a, b et c

fonctions avec paramètres de retour

Une fonction peut retourner une valeur, un objet via le mot clef **return**

Exemple de fonctions sans retour

```
def afficheSomme(a,b)  :
      print('la somme vaut '+(a+b))
```

Exemple de fonctions avec retour

```
def somme(a,b)  :
      return a+b
```

Utilisation de la fonction dans le main

```
>>afficheSomme(3,4)
la somme vaut 7
>>print(afficheSomme(3,4))
la somme vaut 7
None
#cette instruction demande à la fonction somme de calculer le résultat et de l'afficher elle-même comme la fonction en retourne rien, print de « rien » affiche None
```

Utilisation de la fonction dans le main

```
>>somme(3,4)
#cette instruction calcule 7 mais ne l'affiche pas
>>print('la somme vaut '+ somme(3,4))
la somme vaut 7
#cette instruction demande à la fonction somme de calculer le résultat et c'est le main qui affiche le résultat
```

Nombre de paramètres de retour

Via une liste ou un tuple, une fonction python peut retourner plusieurs valeurs

Entrée de la fonction

a vaudra 5
b vaudra 2

Exemple de fonctions retournant un tuple

```
def sommeetmultiplication(a,b)  :
        return ( a+b, a*b)
```

Utilisation de la fonction dans le main

```
>>x=5
>>y=2
>>z,w= sommeetmultiplication(x, y)
>>print(z)
7
>>print(w)
10
```

Sortie de la fonction

z,w = (7,10)

On utilise la multi-affectation et la décompression du tuple pour récupérer les deux valeurs

fonctions lambda ou fonctions anonymes

Une fonction lambda fait la même chose qu'une fonction ordinaire, elle prend un nombre quelconque d'arguments (y compris des arguments optionnels) et retourne la valeur d'une expression unique.

Il n'y a pas de parenthèses autour de la liste d'arguments
Il n'y a pas de mot-clé return car la fonction complète ne pouvant être qu'une seule expression).

Exemple de fonction classique

```
>>> def f(x):
...     return x*2
...
>>> f(3)
6
```

Exemple de fonction lambda assignée à une valeur

```
>>> g = lambda x: x*2  1
#on assigne à g la declaration de la fonction
>>> g(3)
6
```

Exemple de fonction lambda sans assignation

```
>>> (lambda x: x*2)(3) 2
6
```

La fonction est anonyme, elle n'a pas de nom, mais elle peut être appelée à travers la variable à laquelle elle est assignée.

Décorateurs

Un décorateur est une fonction qui prend en paramètre une fonction et qui retourne une fonction.
Ainsi un décorateur est une fonction qui modifie le comportement d'autres fonctions.

Les décorateurs sont utiles lorsque l'on veut ajouter du même code à plusieurs fonctions existantes.

```
def decorator(f):
# decorator est un décorateur
                print(f.__name__)
                 return
```

```
>>> @decorator
... def addition(a, b):
...     return a + b
...
addition
```

Équivalent à

```
def addition(a, b):
    return a + b

addition = decorator(addition)
```

Il est possible à un décorateur d’avoir des paramètres

Créer ses propres modules

- Pour créer un module, il suffit de programmer les fonctions qui le constituent dans un fichier portant le nom du module, suivi du suffixe « .py ».
- Depuis un (autre) programme en Python, il suffit alors d'utiliser la primitive import pour pouvoir utiliser ces fonctions.

- On peut aussi définir des variables globales dans un module.

Dans le fichier monmodule.py

```
maliste=[1,2,3]

def longueur(uneliste):
  return len(uneliste)
```

Dans l'interpréteur

```
>>import monmodule
>>print(monmodule.maliste)
[1,2,3]
>>print(monmodile.longueur(monmodule.maliste))
3
```

On préfixe le nom des variables et des fonctions par le nom du module

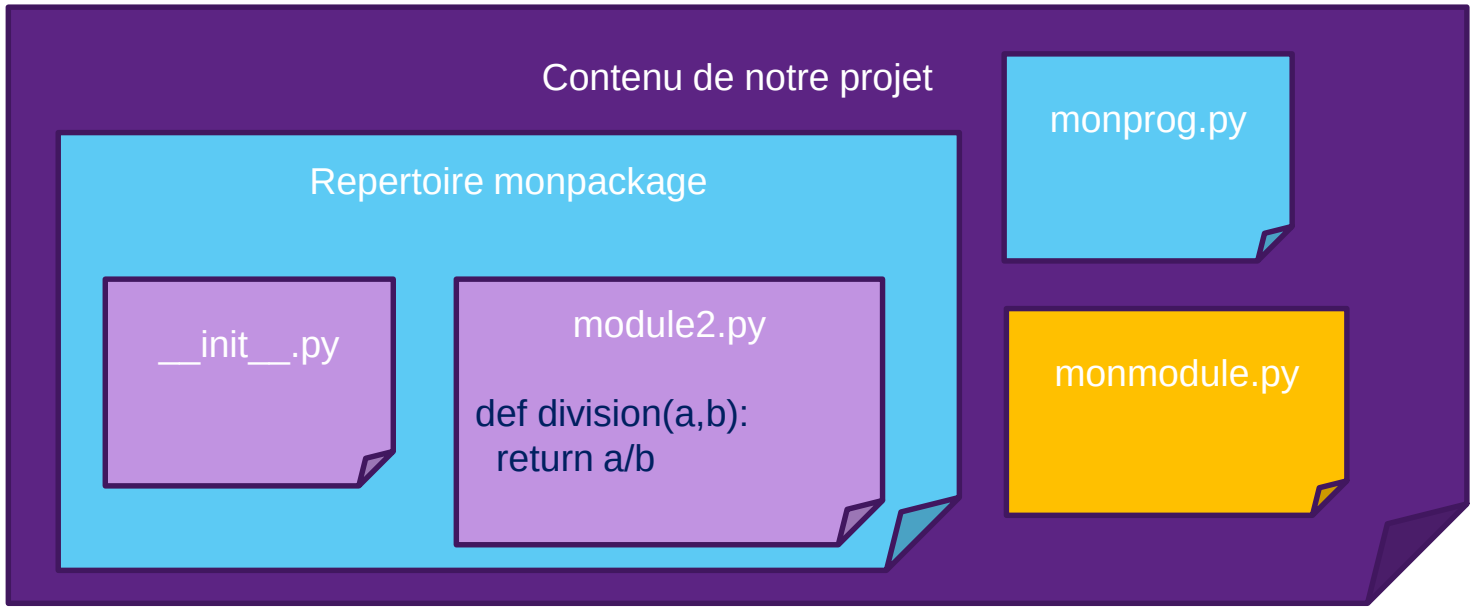
Créer ses propres paquets

- un *package* est un répertoire dans lequel est défini un fichier __init.py__ regroupant un ensemble de *modules* ou de *sous-packages* (organisation en sous-répertoires possible)

Etape 1 : création d'un répertoire qui sera notre package

Etape 2 : Création d'un fichier **__init__.py** , cela indique à python qu'il s'agit d'un **package**. Ce fichier peut être vide, seule sa présence est importante.

Etape 3 : Création d'un module ou plusieurs modules dans notre package



monProg.py

```
>>import monmodule
>>print(monmodule.maliste)
[1,2,3]
>>print(monmodile.longueur(monmodule.maliste))
3
>>from monpackage.module2 import division
#on importe du module venant du package la fonction qui nous interesse
>>print(module2.division(6,3))
2
```

La localisation des *modules* et des *packages* est définie par la variable d'environnement PYTHONPATH

ESIEE[it]
L'école de l'expertise numérique

une école de la
CCI PARIS ILE-DE-FRANCE

www.esiee-it.fr