AGRO CAMPUS OUEST

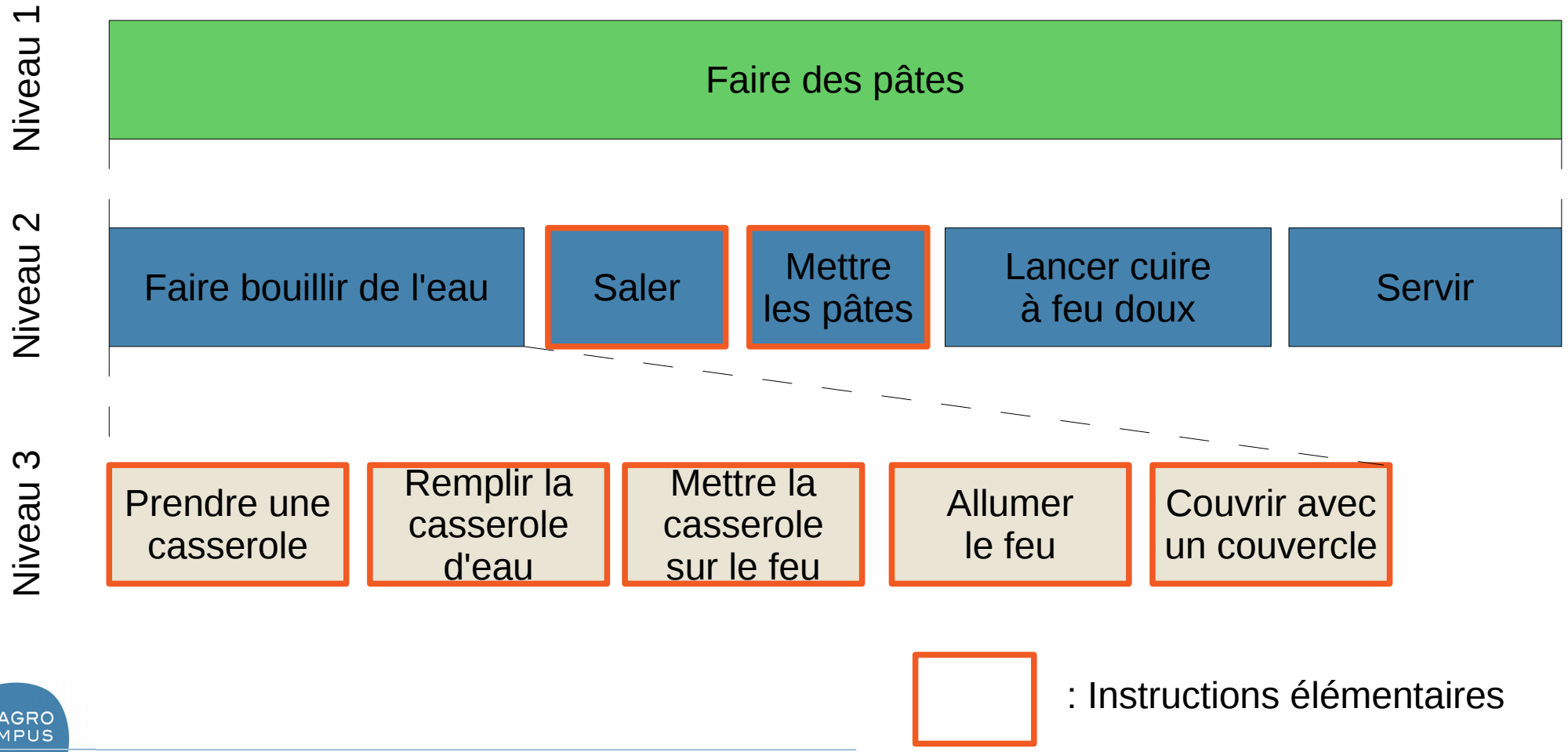
Python 3 – Fonctions

Plan

1) Principes
2) Listes
3) Ensembles
4) Dictionnaires

Principe de l'approche descendante

- Approche descendante = approche systématique du général au particulier



Intérêts de l'approche descendante

- Cadre de réflexion puissant
  - « diviser pour régner »
    - Divise des tâches complexes en sous-tâches simples à programmer
    - La combinaison entre les sous-tâches se fait au travers des structures de contrôles usuelles
  - approche générique (pas uniquement pour l'informatique)
    - Applicable à de nombreuses situations réelles (planification de tâches, rédaction d'une stratégie d'actions, ...)

Intérêts de l'approche descendante

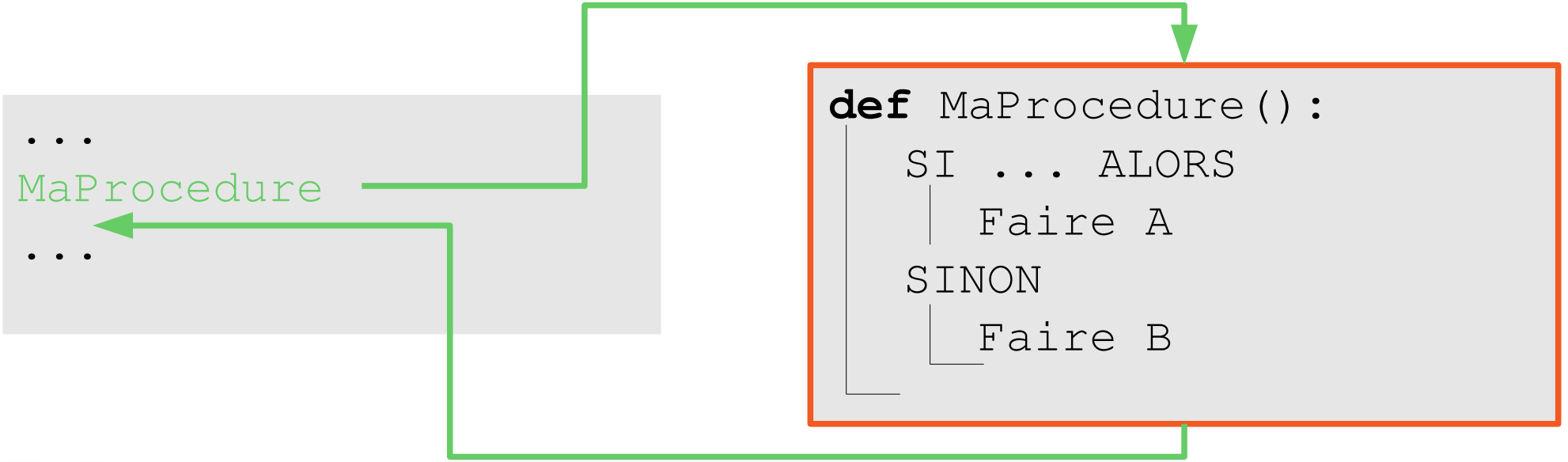
- Bonne pratique de la programmation
  - Réutilisabilité
    - *Si je sais faire bouillir de l'eau pour faire des pâtes, je sais faire bouillir de l'eau pour faire du riz.*
    - L'informatique est une science paresseuse : il faut essayer de réutiliser ce qui a déjà été fait !
  - Sécurité
    - *Si ma procédure pour faire cuire de l'eau tient la route ! Mieux vaut l'utiliser telle quelle plutôt que d'en inventer une autre !*
    - **Moins on a de travail à faire, moins on risque de faire des erreurs**
    - Il faut éviter d'avoir à écrire deux fois la même chose : si on y trouve une erreur, il y a de forte chance qu'on oublie de modifier toutes les copies !

Intérêts de l'approche descendante

- Cadre formateur pour l'apprentissage de la programmation informatique
  - Une sous-tâche usuelle devient une « instruction de base » pour le programmeur
    - L'apprenti-programmeur étend sa gamme d'instructions
    - Il peut réutiliser pour concevoir des algorithmes (comme des instructions de bases)
  - L'apprentissage facilite le travail de conception futur en permettant à l'apprenti de raisonner à des niveaux plus abstraits

Les procédures

- Les **procédures** sont des sous-programmes qui réalisent des sous-tâches
- L'**appel de procédure** est une instruction qui lance l'exécution d'une procédure
- Écriture en pseudo-code :



Écriture d'une procédure en pseudo-code

- Une procédure est un « sous-programme » auquel on peut faire appel dans un autre programme
- Une procédure à des paramètres (ou non !)
  - on parle de **paramètres formels**
  - l'action de la procédure change en fonction de la valeur des paramètres formels

```
def MaProcedure(int x, double f) :
    SI x>0 ALORS
        print x*f
    SINON
        print -x
```

Appel d'une procédure
… avec passage de valeurs

```
def MaProcedure(int x, double f) :
    SI x>0 ALORS
        x <- x*f
    SINON
        x <- -x
    print x
```

Programme principal

```
print 'debut !'
MaProcedure(-5 , 2)
print 'fin … et suite'
```

Affectation des paramètres formels

Exemple de procédure Python

- Table de multiplication

```
def table_par_7():
    nb = 7
    i = 0 # compteur
    while i < 10:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1 # On incrémente i

#debut du programme
Table_par_7()
```

Exemple de procédure Python

- Généralisation
  - Pas de typage des paramètres
  - Principe de « substitution » d'un cas particulier

```
def TableMult(nb):
    i = 0 # compteur
    while i < 10:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1 # On incrémente i

#debut du programme
TableMult(7)
TableMult(9)
```

Exemple de procédure Python

- Quelques détails syntaxiques
  - Possibilité d'utiliser des **paramètres nommés**
  - Utilisation des **paramètres par défaut**

```
def TableMult(nb, maxtable =10):
    i = 0 # compteur
    while i < maxtable:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1 # On incrémente i

#debut du programme
TableMult(5)
TableMult(maxtable=3)
TableMult(maxtable=3, nb=6) #nommage des paramètres
```

Appel d'une procédure

- Les paramètres formels d'une procédure ne peuvent être utilisés uniquement dans la procédure
  - On parle de « visibilité de la variable »
- Les variables sont passées *par valeur*
  - Les paramètres formels sont initialisés avec les valeurs des paramètres effectifs (passés en paramètres de l'appel)
  - Les modifications d'un paramètre formel **n'a pas** d'influence sur la valeur du paramètre effectif.
- En Python, le passage de paramètre se fait par valeur
- Le passage de paramètre par valeur s'oppose à un passage de paramètre « par référence » ou « par pointeur »

Les fonctions : la déclaration

- **Une fonction est une procédure qui *retourne* une valeur**
  - La fonction est typée
  - Lors de l'appel, la fonction *vaut* cette valeur
- L'instruction **`return`** indique, dans la fonction, la valeur que retourne la fonction

```
def max(x, y) :
    mx=x
    if x<y :
        mx = y
    return mx
```

```
X = max(34, 56)
```

Autre exemple en Python

```
import math

def carre(valeur):
    # Fonction de calcul d'un carré de valeur
    # - valeur : nombre réel

    C =  valeur * valeur
    return C

#debut du programme
diffcarre = carre(5.2) - carre(4.2)
diff = math.sqrt( diffcarre )
```