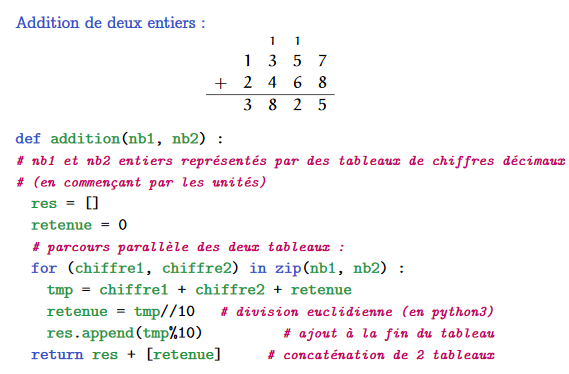
Cours 1

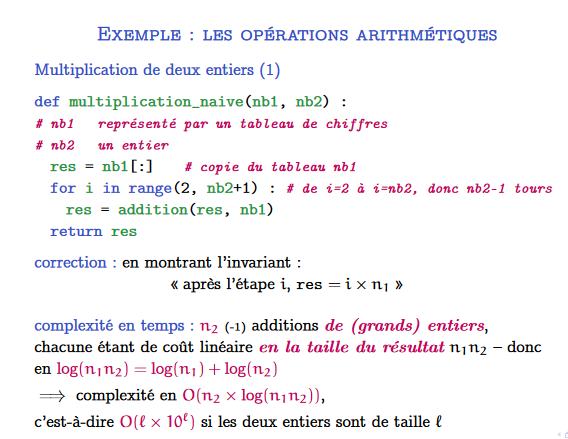
* algorithmique = « conception et analyse des algorithmes»
* algorithme = « méthode (systématique) de résolution d’un problème »

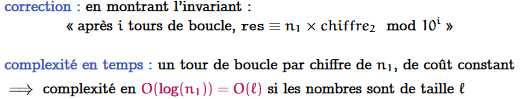
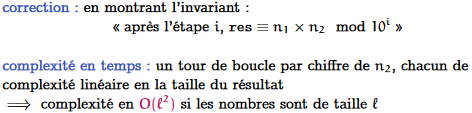
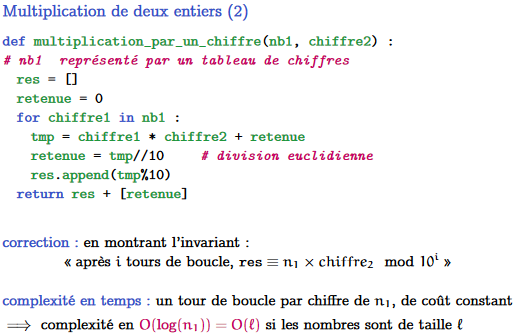
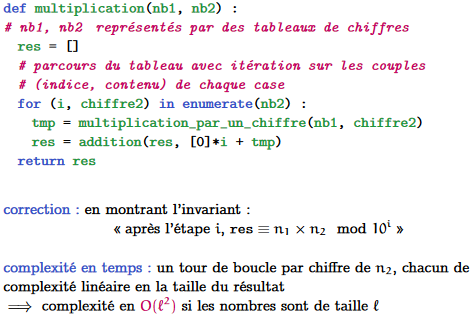
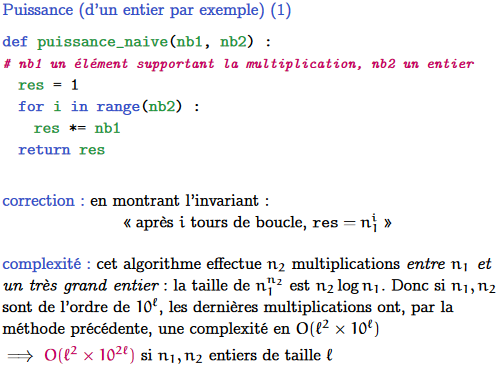
trois axes d’étude :

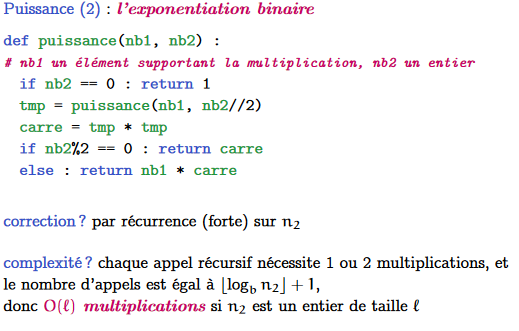
* conception: y a-t-il des techniques générales?
* preuve de correction: un algorithme est correct si, pour chaque entrée, il termine en produisant la bonne sortie
* étude de l’efficacité: les ressources nécessaires (temps, mémoire)sont-elles raisonnables ? Est-il possible de faire mieux ?



**correction** : en montrant l’invariant : « après i tours de boucle, res = n1+n2 % 10^i »

**complexité en temps** : autant d'additions élémentaires que de chiffres dans l’écriture décimale des entiers.  
⇒« complexité linéaire» sous-entendu « en la taille L des données », la taille étant ici le nombre de chiffres décimaux : dire que n1 et n2 sont de taille L signifie que n1,n2 ∈ O(10^L), ou encore que L = 1 + [ max(log10 (n1) , log10 (n2) ) ]





Cours 2

# Fibonacci

| def fibo\_1(n) :  if n <= 0 : return 0 if n <= 2 : return 1 return fibo\_1(n-1) + fibo\_1(n-2) |
| --- |

**Preuve de terminaison** : par récurrence (forte)

* cas de base : si n <= 2, fibo\_1(n) termine
* hérédité : soit n >= 2 t.q. fibo\_1(k) : termine pour tout k < n alors fibo\_1(n−1) et fibo\_1(n−2) terminent, donc fibo\_1(n) termine

donc fibo\_1(n) termine pour tout n

**Preuve de correction** : par récurrence (forte)

* cas de base : sin <= 2, fibo\_1(n) retourne Fn
* hérédité : soit n >= 2 t.q. fibo 1(k) retourne k pour tout k < n alors fibo 1(n) retourne fibo\_1(n−1) + fibo\_1(n−2) = F(n−1) + F(n−2) = Fn

Donc Fibo\_1(n) retourne pour tout n

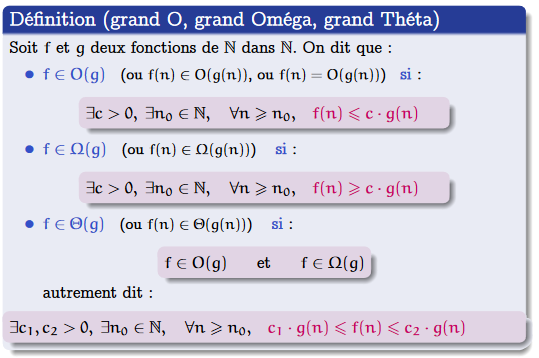
**(gros) inconvénient** : recalcul permanent de valeurs déjà calculées

1. utiliser directement la définition par récurrence
2. garder un tableau de toutes les premières valeurs

| def fibo\_2(n) :  if n <= 0 : return 0 liste = [0, 1] + [0] \* (n-1) for i in range(2, n+1) :  liste[i] = liste[i-1] + liste[i-2]  return liste[n] |
| --- |

1. utiliser directement la définition par récurrence
2. garder un tableau de toutes les premières valeurs
3. garder seulement les deux dernières valeurs

| def fibo\_3(n) :  if n <= 0 : return 0 previous, last = 0, 1 for i in range(2, n+1) :  previous, last = last, previous + last  return last |
| --- |

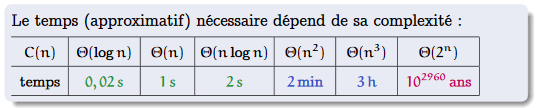


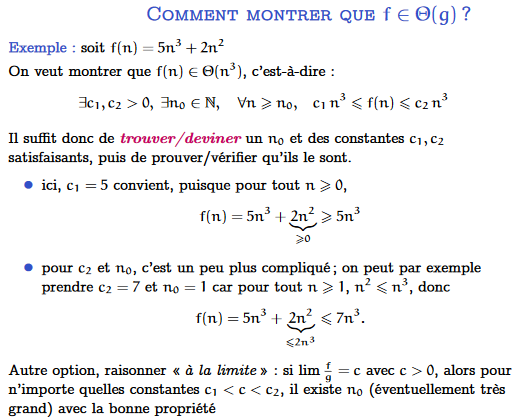
KK

Cours 3

# Complexité et ordres de grandeur

Considérons un algorithme(ou plutôt un programme, dans un langage donné, sur une machine donnée), qui met 1 centième de seconde à traiter les entrées de taille n=100





Cours 4

Cours 5

Cours 6

Cours 7

Cours 8

Cours 9

Cours 10

Cours 11

Cours 12