DUT SRC – IUT de Marne-la-Vallée
23/09/2011
INF120 - Algorithmique

Cours 1
Introduction aux algorithmes

Philippe Gambette

Organisation pratique

- **Contact**

- Courriel : philippe.gambette@gmail.com
  (INF120 doit apparaître dans le sujet du courriel)
- Avant ou après le cours

- **Matériel**

- Ordinateur portable : interdit pendant les TP, a priori inutile en cours et TD.
- Pas de téléphone portable pendant cours/TD/TP

- **Déroulement des enseignements**

- Page web du cours : http://tinyurl.com/INF120-2011S1
- En général, distribution de notes de cours à compléter
- Pause (méritée) de 5 minutes entre deux heures de TD
- Passages au tableau pour les corrections d'exercices en TD

Organisation pratique

- **Notes et devoirs**

- Interrogations QCM en début de cours ou TD
  (signalement des absences pour rattrapage, voir intranet)
- Un devoir maison

- **Note finale**

- Prévision : environ 2/3 “compétences”, environ 1/3 “motivation”
- Compétences : 2/3 devoir final (11 janvier 2012), 1/3 QCM
- Motivation : devoir maison, exercices, TP

- **Exercices supplémentaires d'entraînement**

- Sur demande, par courriel
- Sur demande, possibilité d'organiser une séance d'exercices ou de préparation au devoir final.

Sources

- *Le livre de Java premier langage*, d'A. Tasso
- http://www.pise.info/algo/introduction.htm
- Cours INF120 de J.-G. Luque
- http://serecom.univ-tln.fr/cours/index.php/Algorithmie
- Cours de J. Henriet : http://julienhenriet.olympe-network.com/Algo.html
- http://xkcd.com, http://xkcd.free.fr

Plan des cours du semestre

- Introduction aux algorithmes
- Variables et affectation, type et codage des données
- Les tests et la logique
- Les boucles et leur terminaison, les tableaux
- Les entrées-sorties et les fonctions

Plan du cours 1 – Introduction aux algorithmes

- Introduction aux algorithmes
  - A quoi sert un algorithme ?
  - Enjeux de l'algorithmique
  - Algorithme et programme
  - Composants d'un algorithme
- Variables et affectation

A quoi sert un algorithme ?

- À **décrire les étapes de résolution d'un problème :**

  → de façon structurée et compacte

  → à partir d'opérations de base

  → indépendamment d'un langage de programmation

A quoi sert un algorithme ?

- À décrire les **étapes** de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → indépendamment d'un langage de programmation

“**étapes**” aussi appelées “**pas de l'algorithme**”

A quoi sert un algorithme ?

- À décrire les étapes de **résolution d'un problème** :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → indépendamment d'un langage de programmation

Les **données** du problème en **entrée**

Le **résultat** de sa résolution en **sortie**

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → indépendamment d'un langage de programmation

Méthode de résolution d'un problème :

facile à comprendre

facile à transmettre

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → indépendamment d'un langage de programmation

Méthode de résolution d'un problème :

adaptée aux moyens à disposition

adaptée aux connaissances de celui qui l'utilise

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → **indépendamment d'un langage de programmation**

Méthode de résolution d'un problème :

adaptée pour des problèmes qui se traitent sans ordinateur

compréhensible sans apprendre un langage de programmation

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → **indépendamment d'un langage de programmation**

La "minute culturelle"

Algorithmes **sans ordinateurs** :
- Euclide (vers -300) : calcul du PGCD de 2 nombres

Euclide & Ada Lovelace : Wikipedia
Al-Khuwārizmī : Flickr (Heathen Dawn)

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → **indépendamment d'un langage de programmation**

La "minute culturelle"

Algorithmes **sans ordinateurs** :
- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations

Euclide & Ada Lovelace : Wikipedia
Al-Khuwārizmī : Flickr (Heathen Dawn)

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème :

  → de façon structurée et compacte

  → à partir d'opérations de base

  → **indépendamment d'un langage de programmation**

La "minute culturelle"

Algorithmes **sans ordinateurs** :
- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations
- Ada Lovelace (1842) : calcul des nombres de Bernoulli sur la *machine analytique* de Charles Babbage

Euclide & Ada Lovelace : Wikipedia
Al-Khuwārizmī : Flickr (Heathen Dawn)

La recette des crêpes

Le site le plus simple pour faire la pâte à crêpes !

Oyé oyé, braves gens ! Bienvenue sur le site le plus simple pour faire la pâte à crêpe ! Cette recette facile de pâte à crêpe se transmet de bouches à oreilles et maintenant de Facebook en Facebook pour votre plus grand plaisir ! Vous allez adorer faire des crêpes.

Pâte à crêpe pour 15 crêpes: **50g de beurre, 4 oeufs, 2 cuillères à café de sucre, 1 pincée de sel, 250g de farine et 1/2 litre de lait**

Mettre l'ensemble des ingrédients dans un récipient sauf le beurre

Mélanger avec un fouet jusqu'à obtenir de la pâte liquide et sans grumeaux

Ajouter les 50g de beurre fondu: fondre au micro-onde, ça va plus vite! Pour plus de goût, ajouter de la fleur d'oranger ou du rhum...

Si possible, laisser reposer, puis étaler une dose de pâte dans une poêle chaude préalablement graissée

Laisser cuire à feu doux...

...puis retourner pour laisser cuire l'autre côté

Bon appétit !

La recette des crêpes

Le site le plus simple pour faire la pâte à crêpes !

Oyé oyé, braves gens ! Bienvenue sur le site le plus simple pour faire la pâte à crêpe ! Cette recette facile de pâte à crêpe se transmet de bouches à oreilles et maintenant de Facebook en Facebook pour votre plus grand plaisir ! Vous allez adorer faire des crêpes.

Pâte à crêpe pour 15 crêpes: **50g de beurre, 4 oeufs, 2 cuillères à café de sucre, 1 pincée de sel, 250g de farine et 1/2 litre de lait**

Mettre l'ensemble des ingrédients dans un récipient sauf le beurre

Mélanger avec un fouet jusqu'à obtenir de la pâte liquide et sans grumeaux

Ajouter les 50g de beurre fondu: fondre au micro-onde, ça va plus vite! Pour plus de goût, ajouter de la fleur d'oranger ou du rhum...

Si possible, laisser reposer, puis étaler une dose de pâte dans une poêle chaude préalablement graissée

Laisser cuire à feu doux...

...puis retourner pour laisser cuire l'autre côté

Bon appétit !

La recette des crêpes

Le site le plus simple pour faire la pâte à crêpes !

Oyé oyé, braves gens ! Bienvenue sur le site le plus simple pour faire la pâte à crêpe ! Cette recette facile de pâte à crêpe se transmet de bouches à oreilles et maintenant de Facebook en Facebook pour votre plus grand plaisir ! Vous allez adorer faire des crêpes.

Pâte à crêpe pour 15 crêpes: **50g de beurre, 4 oeufs, 2 cuillères à café de sucre, 1 pincée de sel, 250g de farine et 1/2 litre de lait**

Mettre l'ensemble des ingrédients dans un récipient sauf le beurre

Mélanger avec un fouet jusqu'à obtenir de la pâte liquide et sans grumeaux

Ajouter les 50g de beurre fondu: fondre au micro-onde, ça va plus vite! Pour plus de goût, ajouter de la fleur d'oranger ou du rhum...

Si possible, laisser reposer, puis étaler une dose de pâte dans une poêle chaude préalablement graissée

Laisser cuire à feu doux...

...puis retourner pour laisser cuire l'autre côté

Bon appétit !

L'“algorithme des crêpes”

Ingrédients : beurre, oeufs, sachets de sucre vanillé, farine, lait, sel

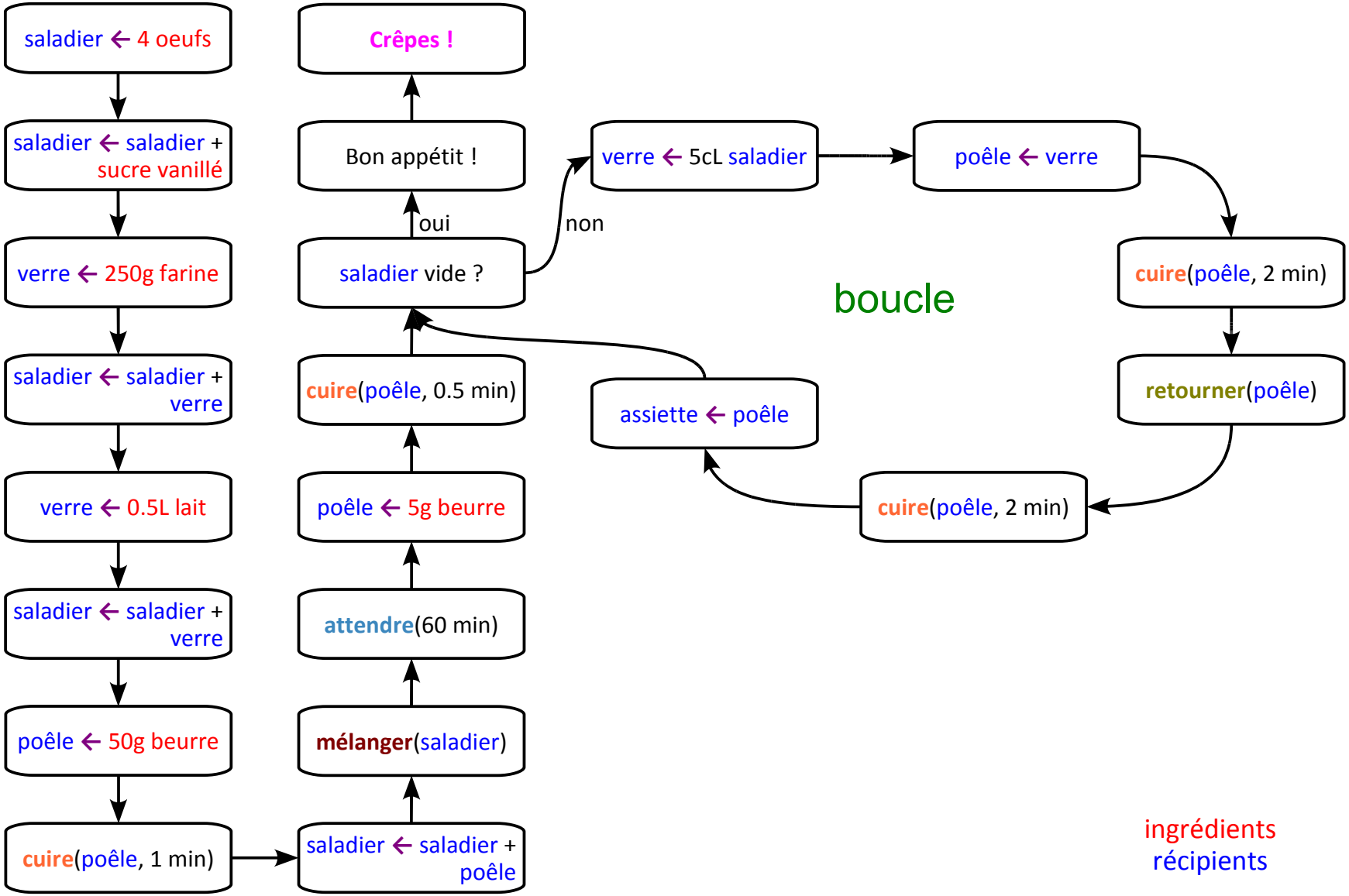
Récipients : saladier, verre mesureur, poêle, assiette

Opérations de base : ***mettre dans un récipient***, ***mélanger***, ***attendre pendant … minutes***, ***retourner***, ***laisser cuire pendant ... minutes***

Algorithme des crêpes :

Mettre 4 oeufs **dans** le saladier
Mettre 1 sachet de sucre vanillé **dans** le saladier
Mettre 250 g de farine **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 0,5 litre de lait **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 50 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 1 **minute**
Mettre le contenu de la poêle **dans** le saladier
Mélanger le contenu du saladier
Attendre pendant 60 **minutes**
Mettre 5 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 0.5 **minute**
Tant que le saladier n'est pas vide :
  Mettre 5 cL du contenu du saladier **dans** le verre mesureur
  Mettre le contenu du verre mesureur **dans** la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Retourner le contenu de la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Mettre le contenu de la poêle **dans** l'assiette

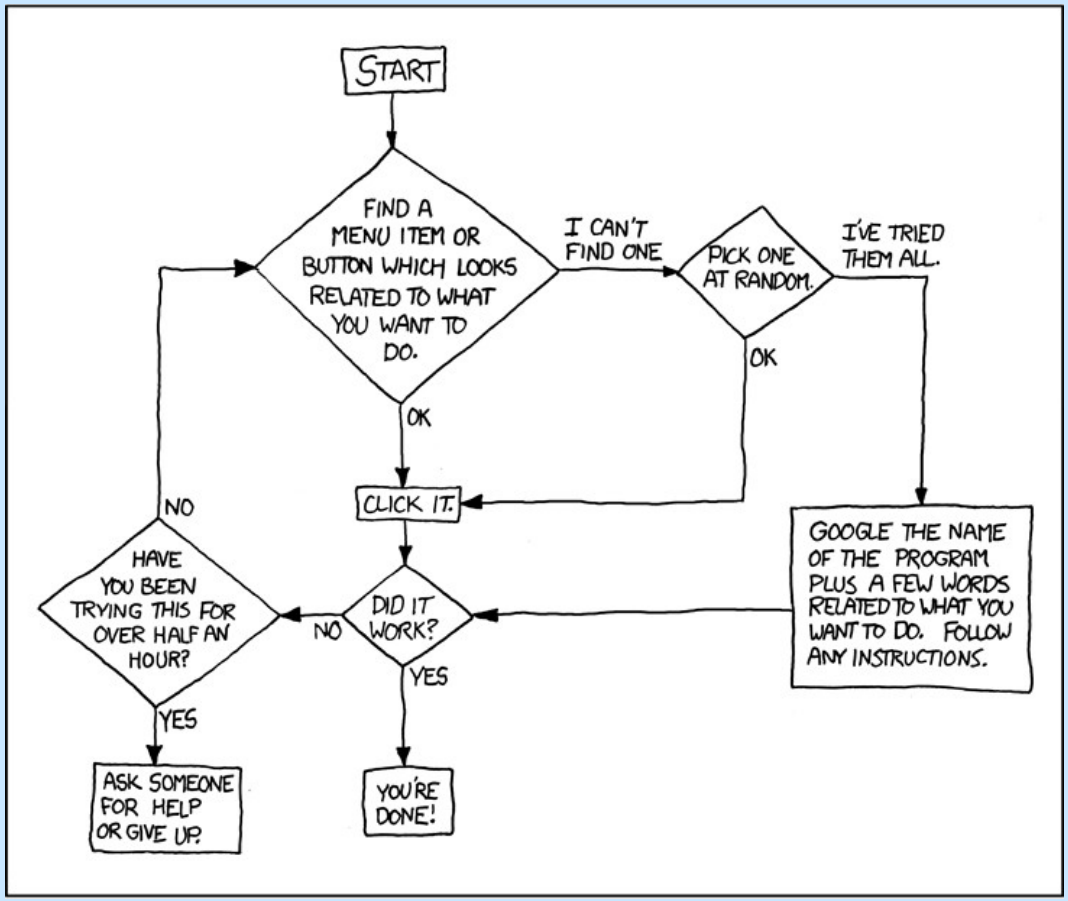
Organigramme de la recette des crêpes



Organigramme de résolution de tout problème logiciel

La “minute xkcd”

Chers parents, grands parents, collègues, et autres non-informaticiens variés.
Nous ne savons pas tout faire dans tous les logiciels comme par magie.
Quand on vous aide, en général on ne fait que ça :



http://xkcd.com/627
http://xkcd.free.fr?id=627

Merci d'imprimer cet organigramme et de le scotcher à côté de votre écran. Félicitations, vous êtes maintenant l'expert du coin en informatique !

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

- critère qui diminue à chaque exécution d'une boucle de l'algorithme

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

- critère qui diminue à chaque exécution d'une boucle de l'algorithme

→ quantité de pâte à crêpes

Algorithme des crêpes :

Mettre 4 oeufs **dans** le saladier
Mettre 1 sachet de sucre vanillé **dans** le saladier
Mettre 250 g de farine **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 0,5 litre de lait **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 50 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 1 **minute**
Mettre le contenu de la poêle **dans** le saladier
Mélanger le contenu du saladier
Attendre pendant 60 **minutes**
Mettre 5 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 0.5 **minute**
Tant que le saladier n'est pas vide :
  Mettre 5 cL du contenu du saladier **dans** le verre mesureur
  Mettre le contenu du verre mesureur **dans** la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Retourner le contenu de la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Mettre le contenu de la poêle **dans** l'assiette

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

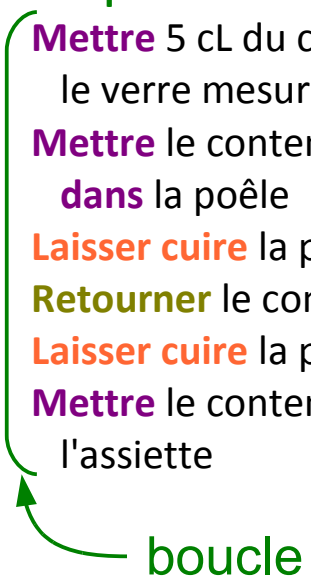
Preuve de terminaison :

- critère qui diminue à chaque exécution d'une boucle de l'algorithme

→ quantité de pâte à crêpes

Algorithme des crêpes :

Mettre 4 oeufs **dans** le saladier
Mettre 1 sachet de sucre vanillé **dans** le saladier
Mettre 250 g de farine **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 0,5 litre de lait **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 50 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 1 **minute**
Mettre le contenu de la poêle **dans** le saladier
Mélanger le contenu du saladier
Attendre pendant 60 **minutes**
Mettre 5 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 0.5 **minute**
Tant que le saladier n'est pas vide :
  Mettre 5 cL du contenu du saladier **dans** le verre mesureur
  Mettre le contenu du verre mesureur **dans** la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Retourner le contenu de la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Mettre le contenu de la poêle **dans** l'assiette



Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

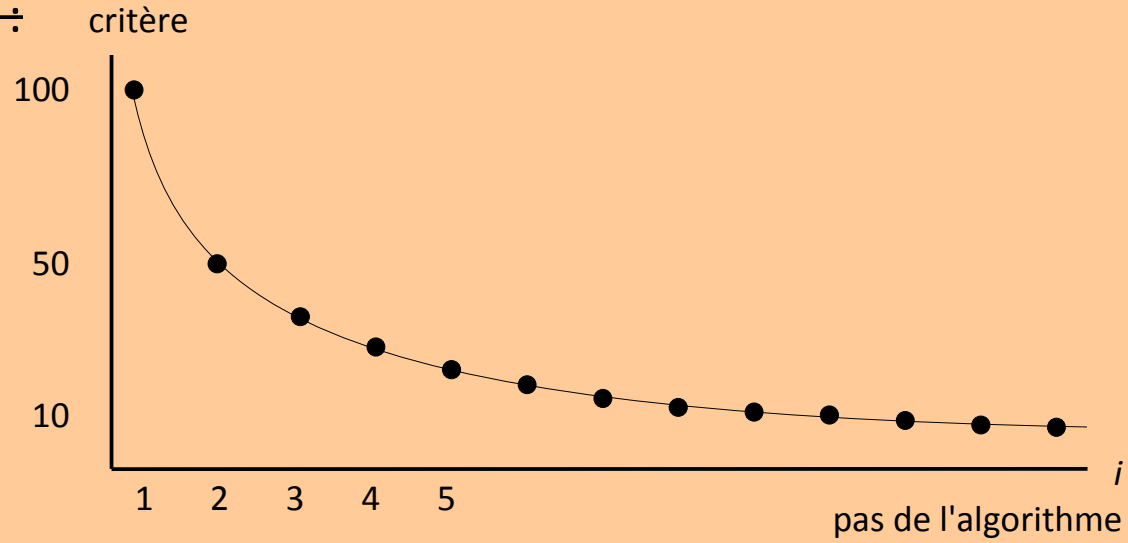
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Valeurs entières :~~

au pas *i*,
critère = 100/*i*



Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

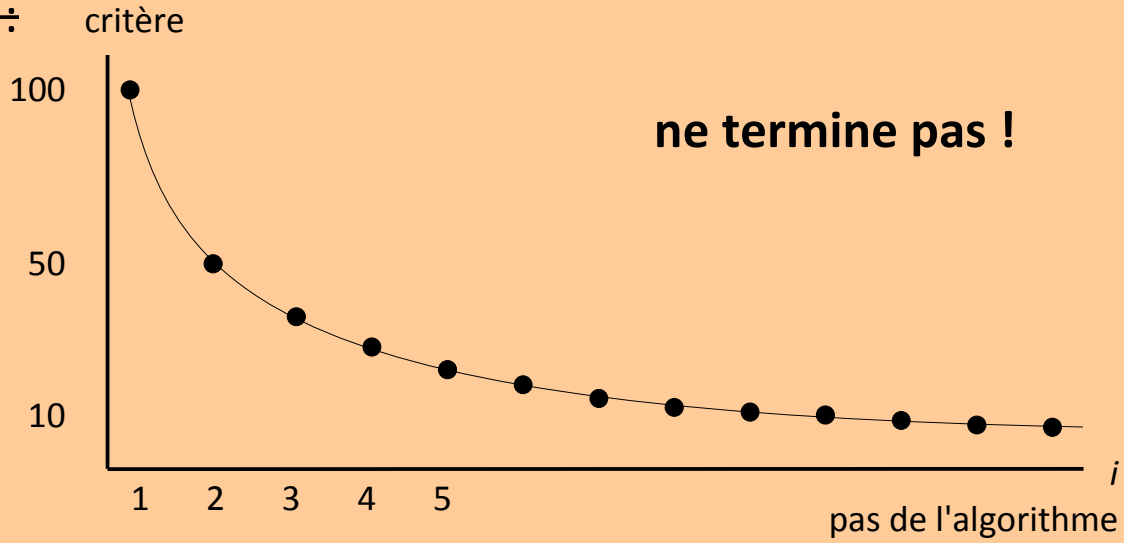
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Valeurs entières :~~

au pas *i*,
critère = 100/*i*



ne termine pas !

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

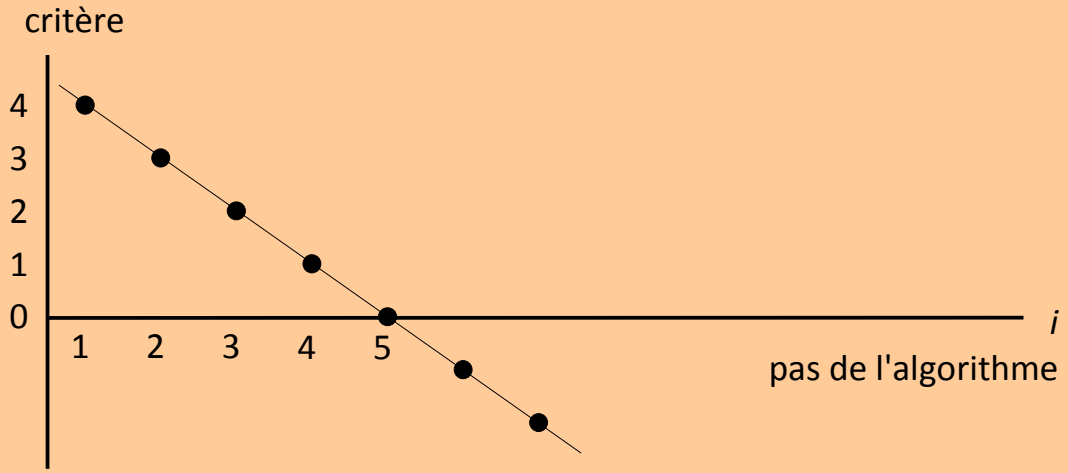
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Valeurs positives :~~

au pas *i*,
critère = 5-*i*



Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

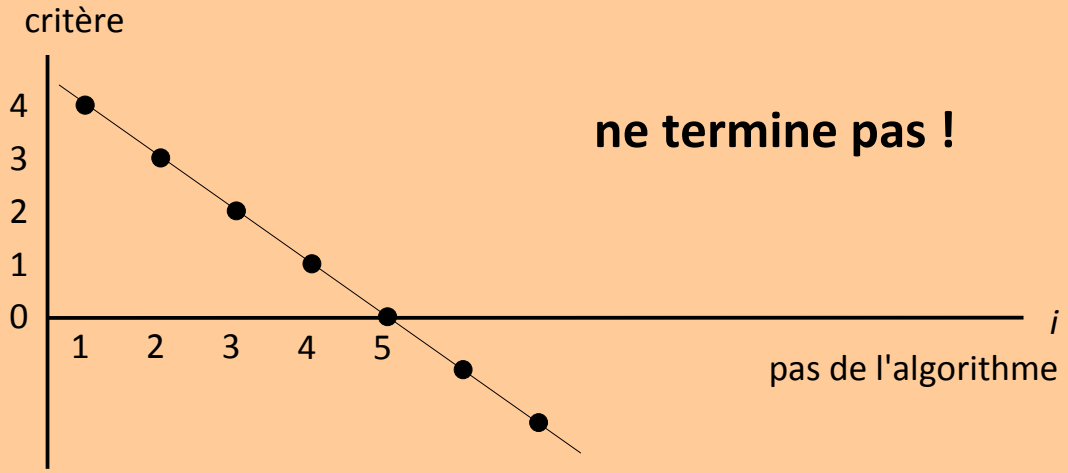
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Valeurs positives :~~

au pas *i*,
critère = 5-*i*



ne termine pas !

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

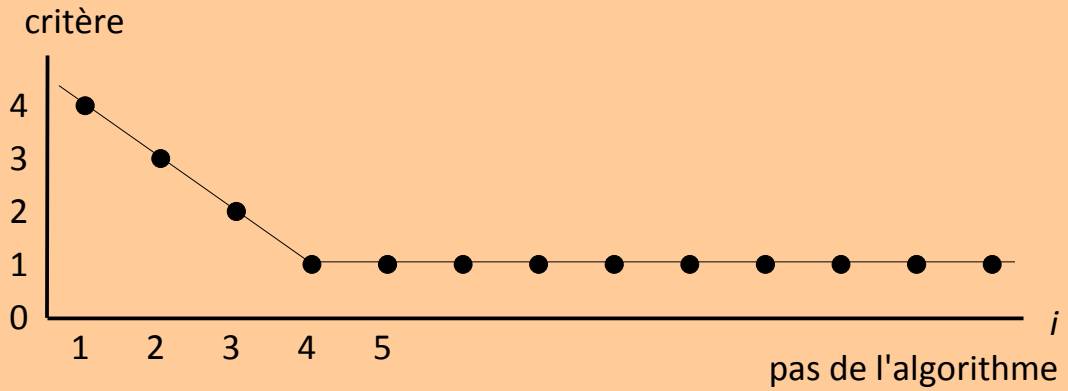
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Strictement décroissante :~~

au pas *i*,
critère =
- 5-*i* si *i*<5
- 1 sinon



Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

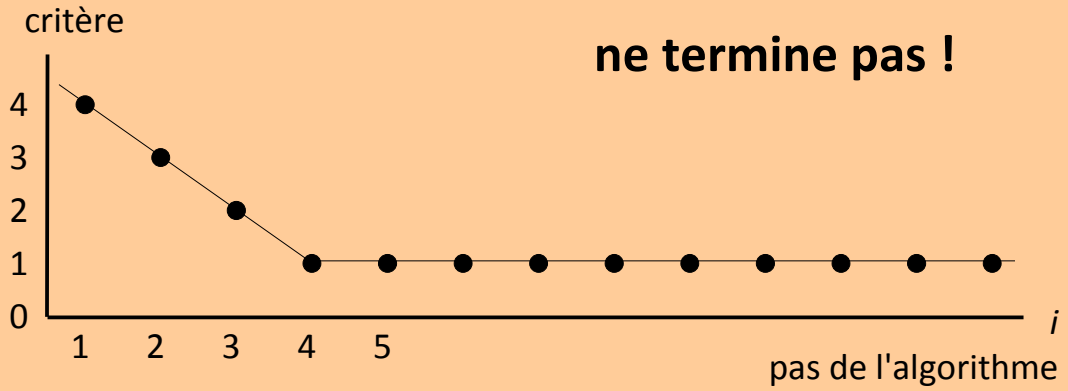
~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

~~Strictement décroissante :~~

au pas *i*,
critère =
- 5-*i* si *i*<5
- 1 sinon



ne termine pas !

Enjeux de l'algorithmique - terminaison

L'algorithme se termine-t-il en un temps fini ?

Preuve de terminaison :

~~- critère qui diminue à chaque exécution d'une boucle de l'algorithme~~

- suite **à valeurs entières positives, strictement décroissante** à chaque exécution d'une boucle de l'algorithme

La "minute mathématique"

Théorème :

Toute suite à valeurs entières positives, strictement décroissante, ne peut prendre qu'un nombre fini de valeurs.

Application :
Trouver la suite qui convient pour prouver la terminaison d'un algorithme.

Enjeux de l'algorithmique - terminaison

La “minute votes SMS”

Problème : aller en voiture de Châtelet à la Tour Montparnasse

Enjeux de l'algorithmique - terminaison

La “minute votes SMS”

Problème : aller en voiture de Châtelet à la Tour Montparnasse

Algorithme “du repère visuel” :

A tout instant on sait où se trouve la Tour Montparnasse

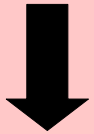
→ prendre la rue qui s'en rapproche le plus

Enjeux de l'algorithmique - terminaison

La “minute votes SMS”

Problème : aller en voiture de Châtelet à la Tour Montparnasse
Algorithme “du repère visuel” :
A tout instant on sait où se trouve la Tour Montparnasse
→ prendre la rue qui s'en rapproche le plus

algorithme de la famille des
algorithmes gloutons



toujours choisir le **profit maximum !**

Enjeux de l'algorithmique - terminaison

La “minute votes SMS”

Problème : aller en voiture de Châtelet à la Tour Montparnasse
Algorithme “du repère visuel” :
A tout instant on sait où se trouve la Tour Montparnasse
→ prendre la rue qui s'en rapproche le plus

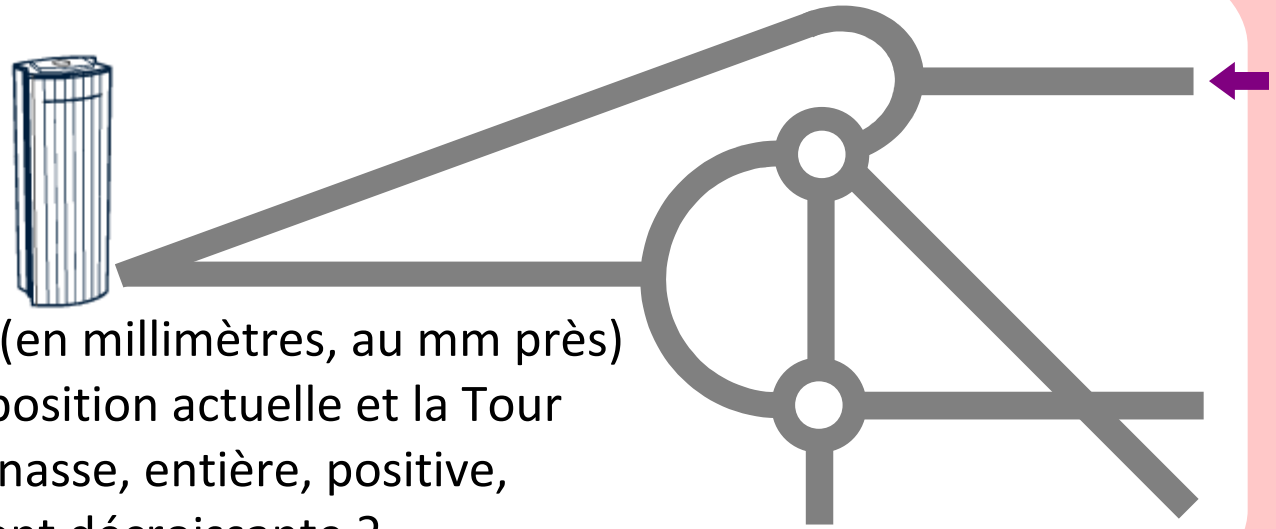
Question : l'algorithme “du repère visuel” termine ?

Enjeux de l'algorithmique - terminaison

La “minute votes SMS”

Problème : aller en voiture de Châtelet à la Tour Montparnasse
Algorithme “du repère visuel” :
A tout instant on sait où se trouve la Tour Montparnasse
→ prendre la rue qui s'en rapproche le plus

Question : l'algorithme “du repère visuel” termine ?



distance (en millimètres, au mm près) entre la position actuelle et la Tour Montparnasse, entière, positive, strictement décroissante ?

Illustration : http://www.tourmontparnasse56.com

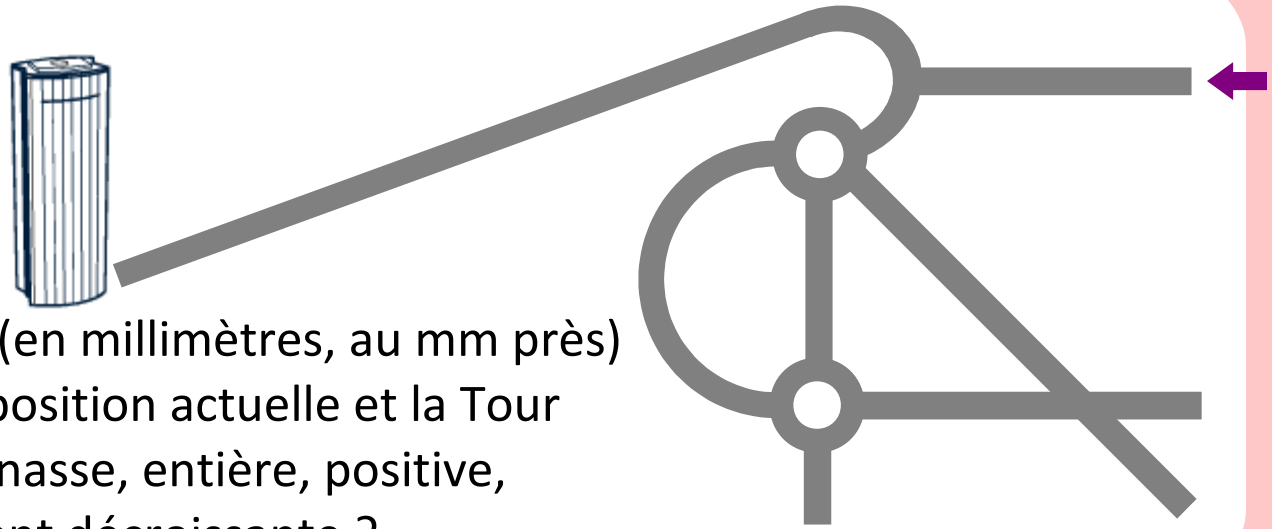
Enjeux de l'algorithmique - terminaison

La "minute votes SMS"

Problème : aller en voiture de Châtelet à la Tour Montparnasse
Algorithme "du repère visuel" :
A tout instant on sait où se trouve la Tour Montparnasse
→ prendre la rue qui s'en rapproche le plus

Question : l'algorithme "du repère visuel" termine ?

NON !



distance (en millimètres, au mm près) entre la position actuelle et la Tour Montparnasse, entière, positive, strictement décroissante ?

Illustration : http://www.tourmontparnasse56.com

Enjeux de l'algorithmique - correction

Correction : L'algorithme donne-t-il le résultat attendu ?

Preuve de correction :

- « invariant » : propriété vraie tout au long de l'algorithme
  - vraie à la première étape
  - si vraie à une étape, vraie à l'étape suivante
  - ⇨ vrai à la fin

En pratique, pour débuter :

- vérifier sur les “cas de base”
- vérifier sur des exemples aléatoires

Enjeux de l'algorithmique - complexité

Complexité : Combien de temps l'algorithme prend-il pour se terminer ?

Théorie de la complexité :
- évaluer le nombre d'opérations en fonction de la taille du problème, dans le pire cas
- prouver qu'on ne peut pas utiliser moins d'opérations pour résoudre le problème, dans le pire cas

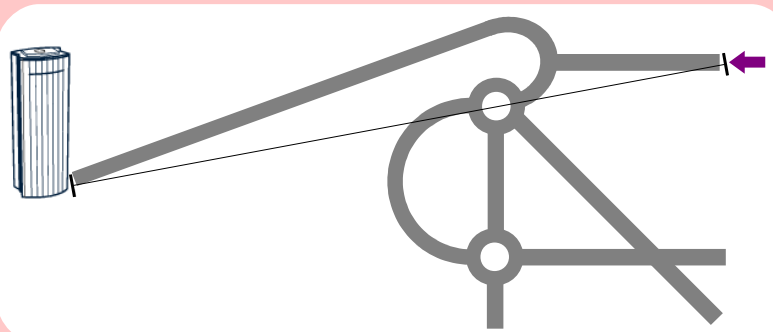
En pratique, pour débuter **:**
- vérifier sur des exemples aléatoires
- connaître les cas difficiles

Enjeux de l'algorithmique - complexité

Complexité : Combien de temps l'algorithme prend-il pour se terminer ?

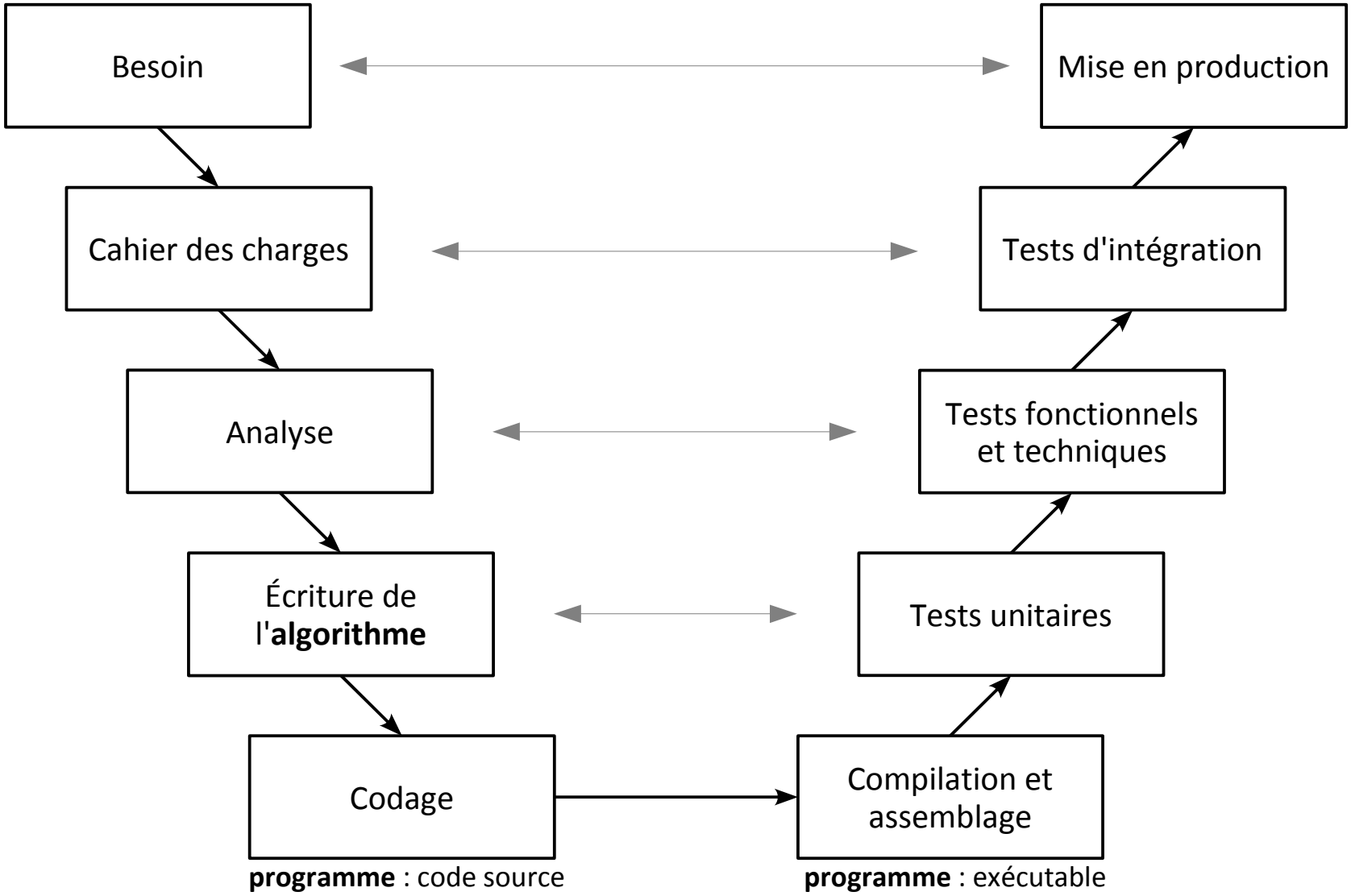
Théorie de la complexité :
- évaluer le nombre d'opérations en fonction de la taille du problème, dans le pire cas
- prouver qu'on ne peut pas utiliser moins d'opérations pour résoudre le problème, dans le pire cas

En pratique, pour débuter **:**
- vérifier sur des exemples aléatoires
- connaître les cas difficiles

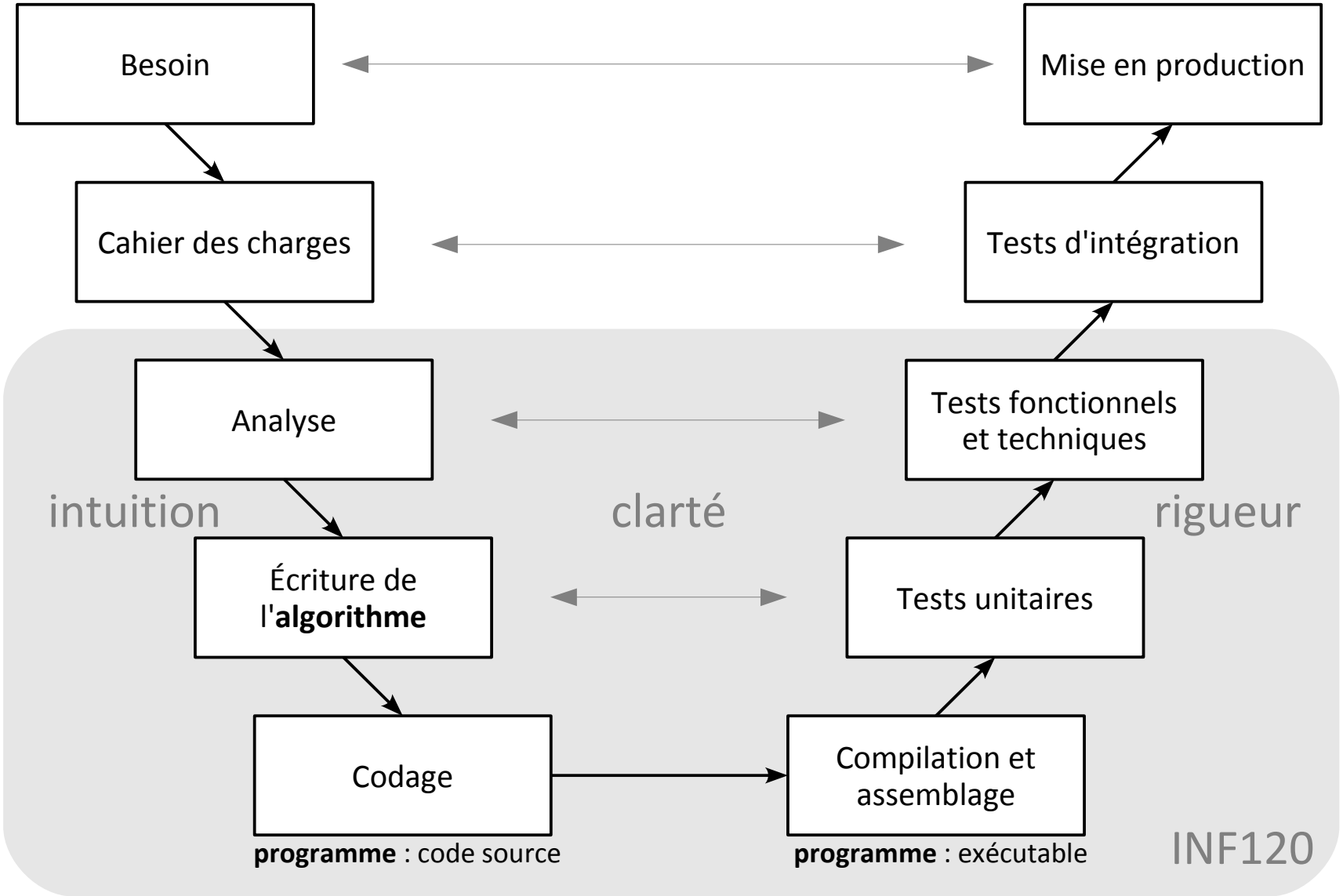


impossible de faire mieux que la ligne droite !

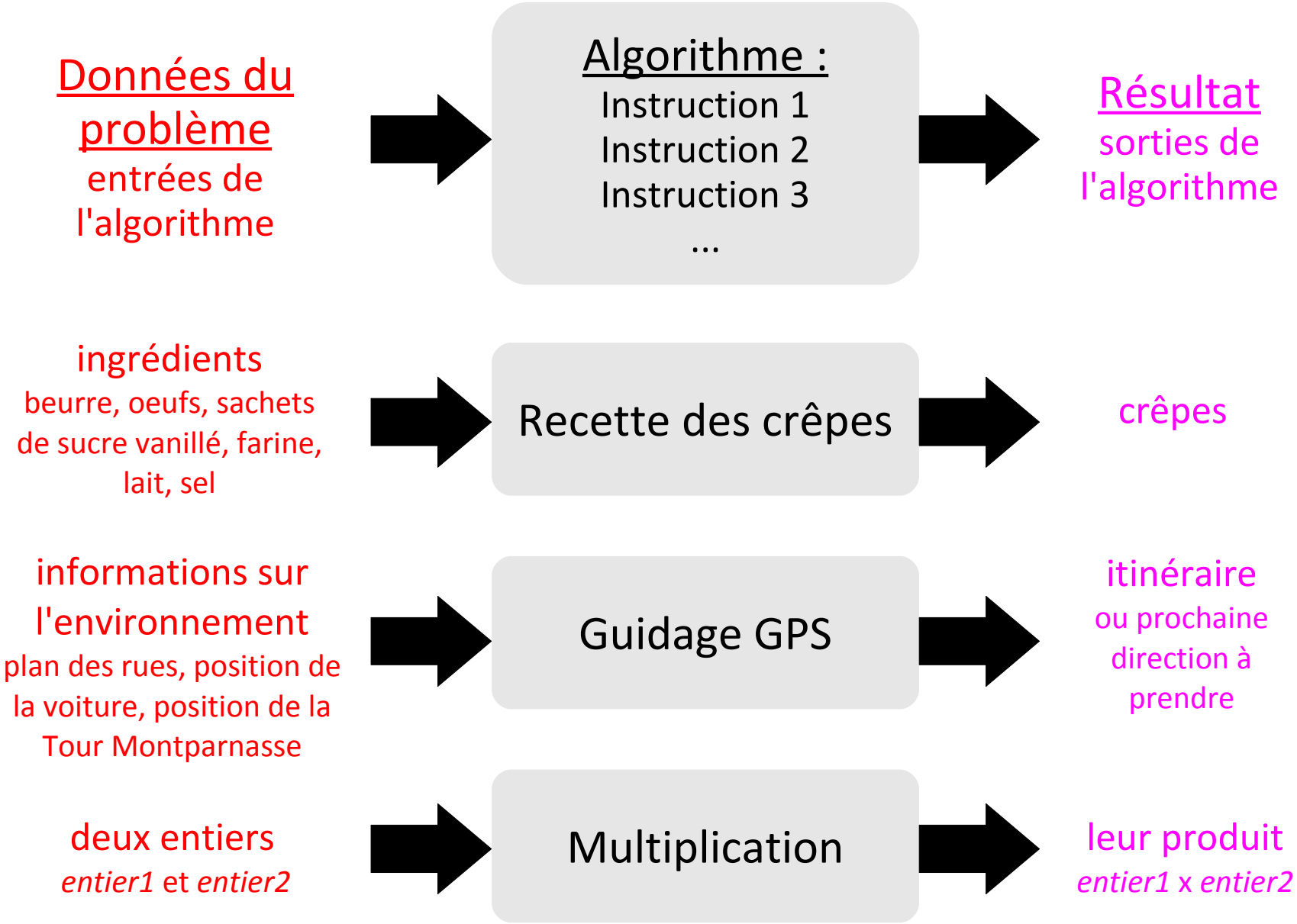
De l'algorithme au programme



De l'algorithme au programme



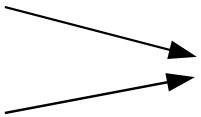
Composants d'un algorithme



Quels types d'instructions ?

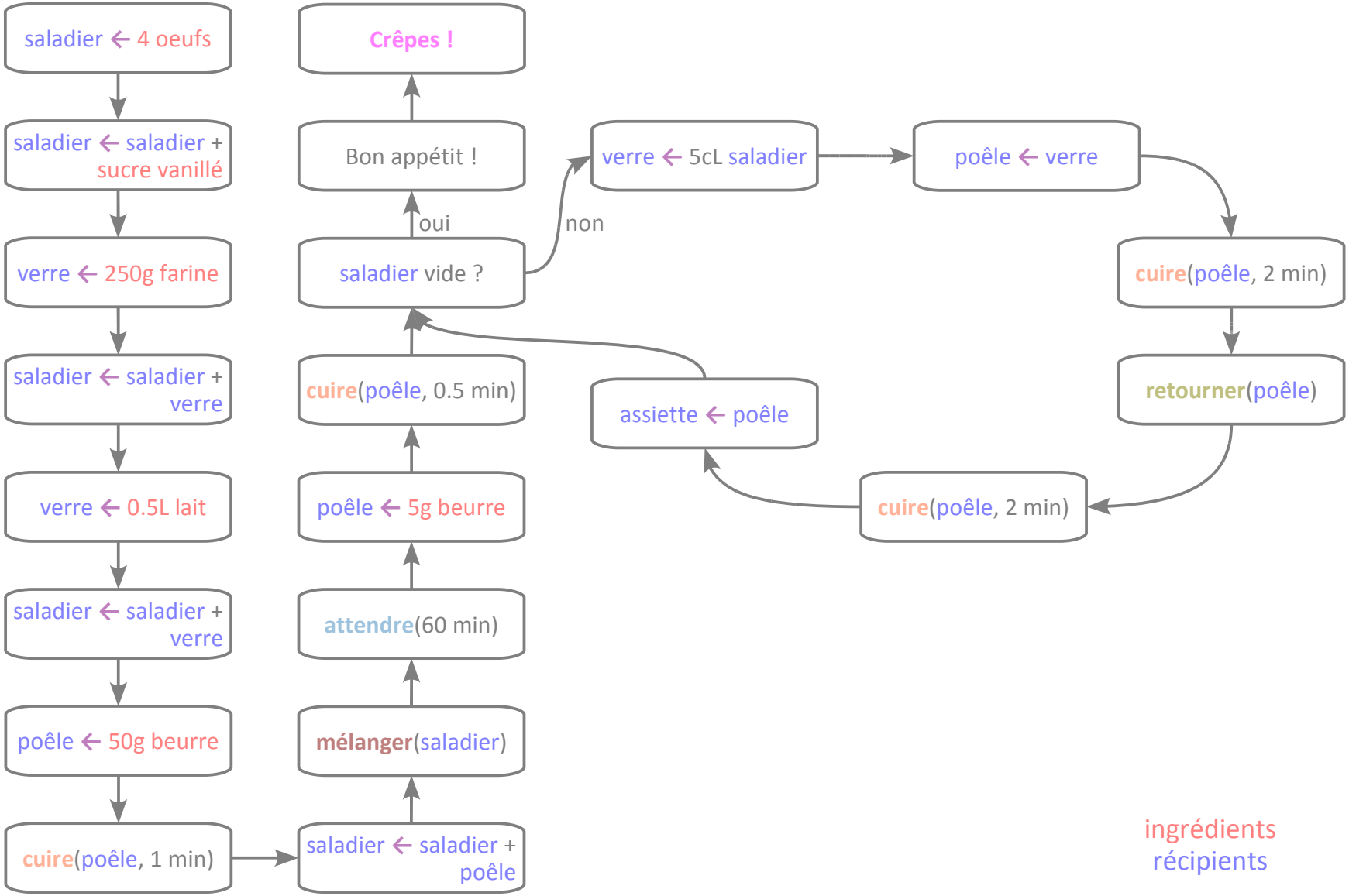
Divers types d'instructions :

- déclaration d'un algorithme
- appel d'un algorithme
- déclaration d'une variable
- affectation d'une variable
- entrées / sorties
- boucle
- test

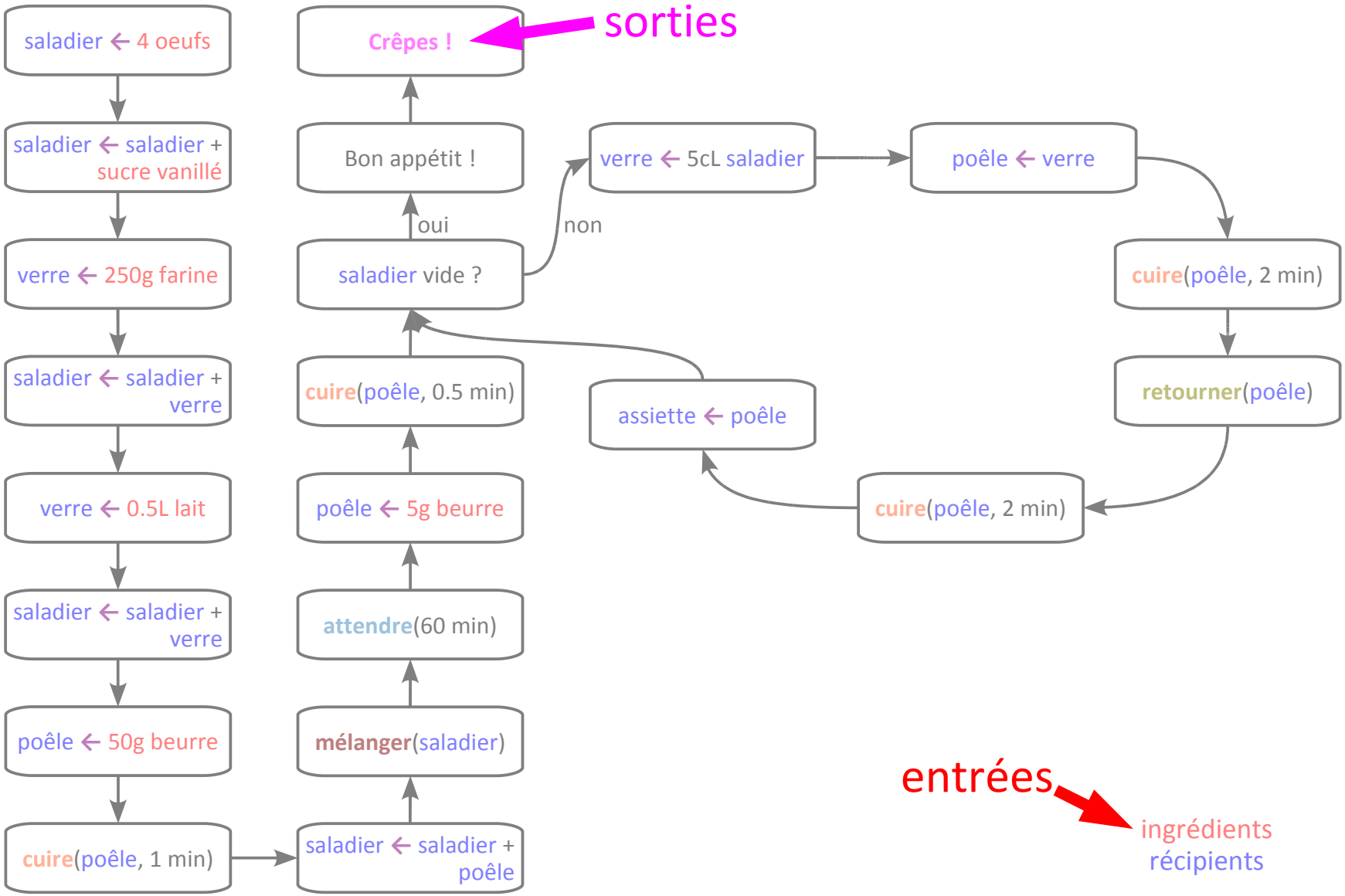


pour stocker des valeurs, des résultats intermédiaires

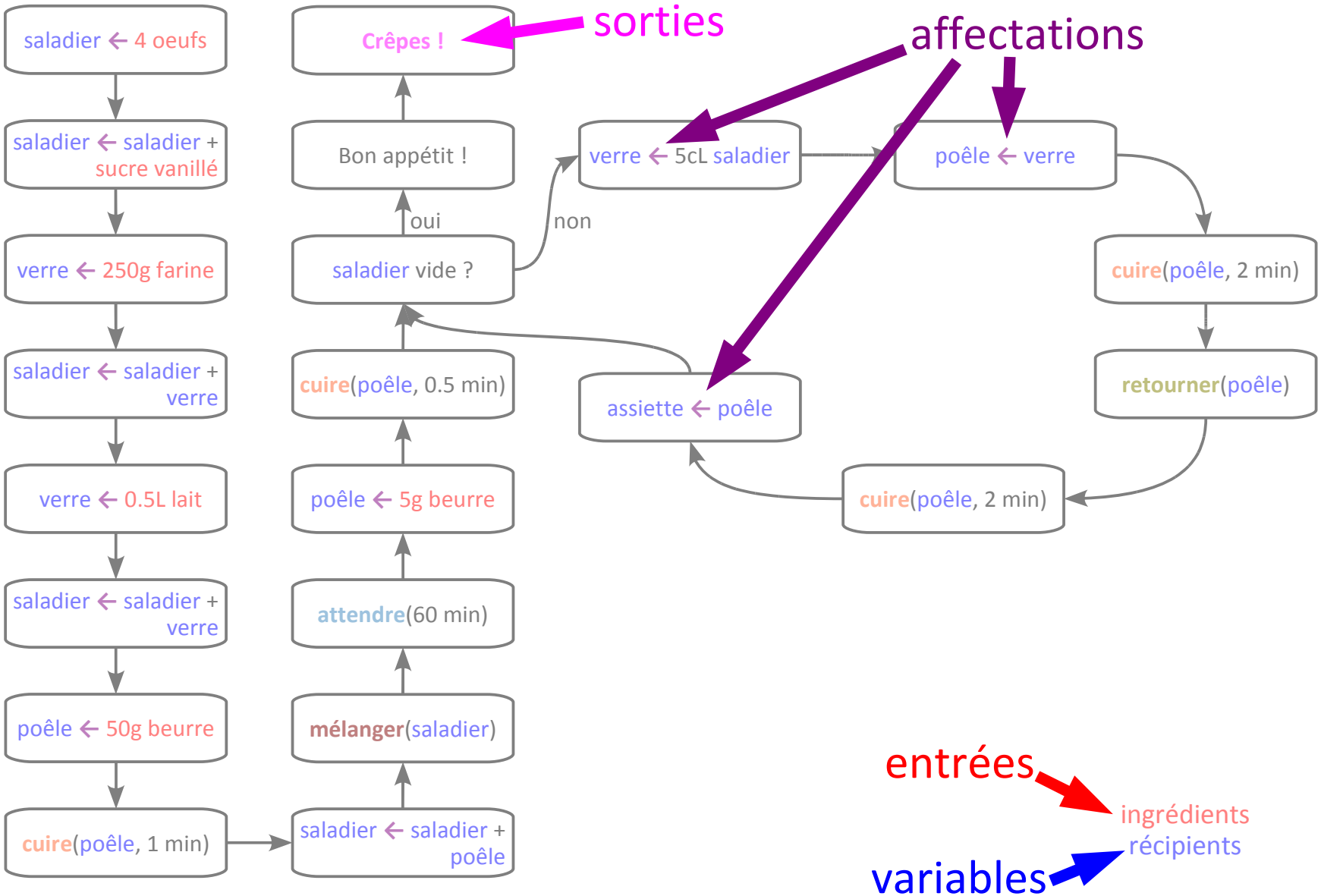
Organigramme de la recette des crêpes



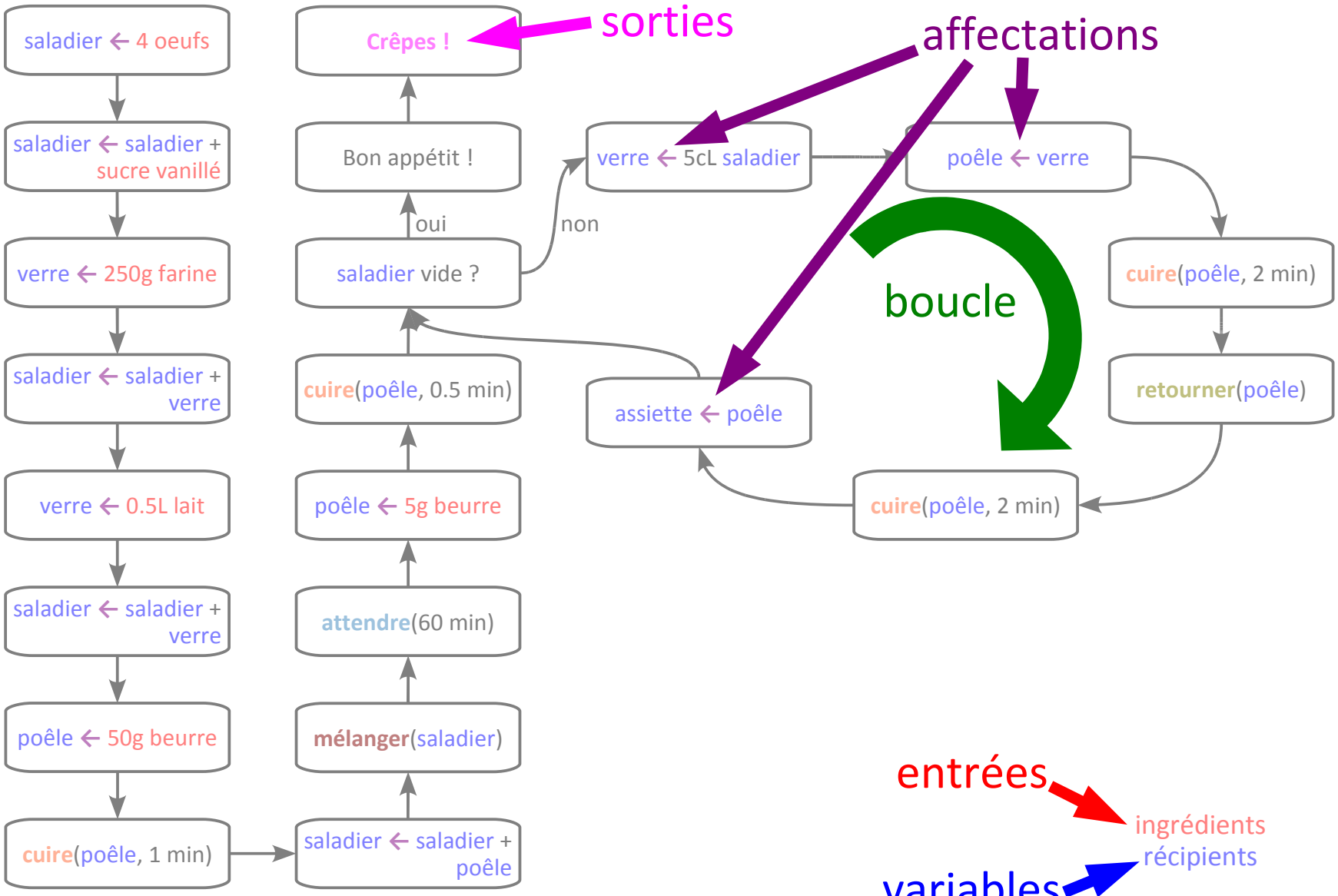
Organigramme de la recette des crêpes



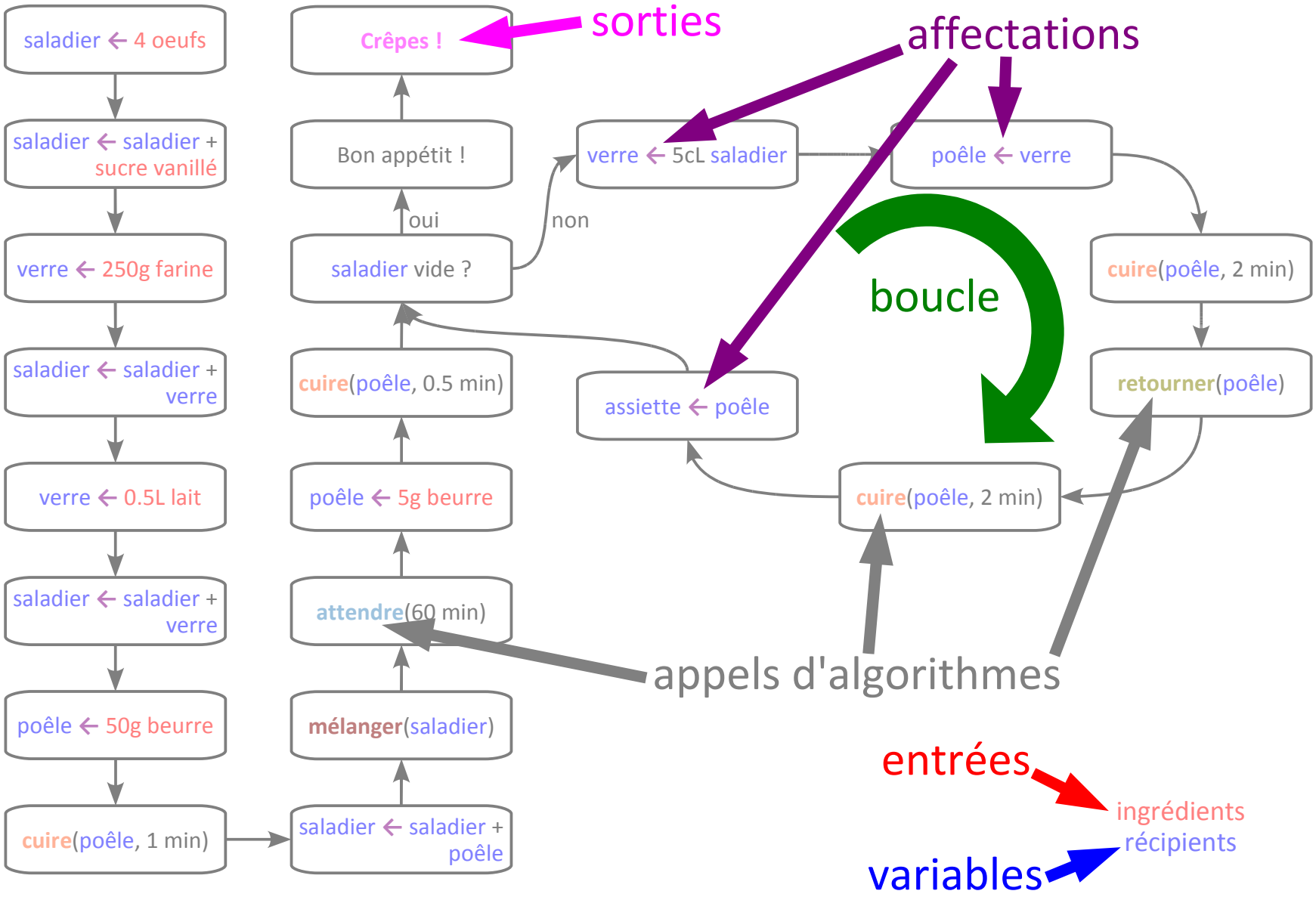
Organigramme de la recette des crêpes



Organigramme de la recette des crêpes



Organigramme de la recette des crêpes



Variables et affectation

Dans un algorithme, une **variable** possède :

- un **nom**,
- une **valeur**,
- un **type** (ensemble des valeurs que peut prendre la variable).

Variables et affectation

Dans un algorithme, une **variable** possède :

- un **nom**,
- une **valeur**,
- un **type** (ensemble des valeurs que peut prendre la variable).

La **valeur** d'une variable :

- est **fixe à un moment donné**,
- peut **changer au cours du temps**.

En revanche, le nom et le type d'une variable ne changent pas.

Variables et affectation

Dans un algorithme, une **variable** possède :

- un **nom**,
- une **valeur**,
- un **type** (ensemble des valeurs que peut prendre la variable).

La **valeur** d'une variable :

- est **fixe à un moment donné**,
- peut **changer au cours du temps**.

L'**affectation** change la valeur d'une variable :

- $a \leftarrow 5$:
  - la variable *a* prend la valeur 5
  - la valeur précédente est perdue (“écrasée”)
- $a \leftarrow b$:
  - la variable *a* prend la valeur de la variable *b*
  - la valeur précédente de *a* est perdue (“écrasée”)
  - la valeur de *b* n'est pas modifiée
  - *a* et *b* doivent être de même type (ou de type compatible)

Variables et affectation

Dans un algorithme, une **variable** possède :

- un **nom**,
- une **valeur**,
- un **type** (ensemble des valeurs que peut prendre la variable).

La **valeur** d'une variable :

- est **fixe à un moment donné**,
- peut **changer au cours du temps**.

L'**affectation** change la valeur d'une variable :

- $a \leftarrow 5$:
  - la variable *a* prend la valeur 5
  - la valeur précédente est perdue (“écrasée”)
- $a \leftarrow b$:
  - la variable *a* prend la valeur de la variable *b*
  - la valeur précédente de *a* est perdue (“écrasée”)
  - la valeur de *b* n'est pas modifiée
  - *a* et *b* doivent être de même type

(ou de type compatible)

La recette de cuisine avec récipients n'est qu'une métaphore

Noms des variables

Dans un **algorithme**, choisir pour les variables :

- un nom composé de **lettres** et éventuellement de **chiffres**
- un nom **expressif**, par exemple :
  - *chaine*, *requête1...* pour une chaîne de caractères
  - *n*, *a*, *b*, *compteur*, *nbOperations*, *longueur...* pour un entier
  - *x*, *y*, *température* pour un réel
  - *estEntier*, *testEntier*, *trouvé...* pour un booléen
- un nom **assez court** (il faut l'écrire !)
- éviter les **noms réservés** : *pour*, *tant que*, *si*...

Dans un **programme** :

- **éviter** les lettres accentuées et la ponctuation
- préférer l'**anglais** si votre code source est diffusé largement
- être **expressif** et **lisible** :
  - *est_entier* ou *estEntier* plutôt que *estentier*

Votre code sera relu, par vous ou par d'autres...

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Intuition :

$$5 \times 3 = \underbrace{5 + 5 + 5}_{3 \text{ fois}}$$

$$\textit{entier1} \times \textit{entier2} = \underbrace{\textit{entier1} + \textit{entier1} + \textit{entier1} + \ldots + \textit{entier1}}_{\textit{entier2} \text{ fois}}$$

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables :

Début

Fin

pseudo-code

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        résultat ← addition(résultat, entier1)
    Fin tant que
    renvoyer produit
Fin
```

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        résultat ← addition(résultat, entier1)
    Fin tant que
    renvoyer produit
Fin
```

Terminaison ?

La condition de sortie de boucle n'est jamais vérifiée car *compteur* ne varie pas.

L'algorithme **ne termine pas**.

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        résultat ← addition(résultat, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        résultat ← addition(résultat, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Terminaison ?

Oui car *compteur* augmente progressivement jusqu'à arriver à *entier2*.

Plus formellement, l'ensemble des valeurs successives de (*entier2* – *compteur*) (à la fin de chaque boucle) est une suite d'entiers positifs strictement décroissante.

L'algorithme termine.

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        résultat ← addition(résultat, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Correction ?

Essayons avec l'exemple :
entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

i	1	2	3	4
compteur	1	2	3	4
résultat	5	10	15	20
entier1	5	5	5	5
entier2	3	3	3	3

La boucle n'est exécutée que 3 fois mais on renvoie 20 : l'algorithme **n'est pas correct !**

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *résultat*

```
Début
    compteur ← 0
    résultat ← entier1
    compteur ← 1
    Tant que compteur < entier2 faire :
        résultat ← addition(résultat, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Correction ?

Essayons avec l'exemple :
entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

i	1	2	3	4
compteur	1	2	3	-
résultat	5	10	15	-
entier1	5	5	5	-
entier2	3	3	3	-

La boucle n'est exécutée que 2 fois et on renvoie 15 : l'algorithme semble **correct...**

... mais ne l'est pas pour *entier2*=0

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← 0
    Tant que compteur < entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← 0
    Tant que compteur < entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Correction ?

entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

i	1	2	3	4
compteur	0	1	2	3
produit	0	5	10	15
entier1	5	5	5	5
entier2	3	3	3	3

On remarque que *produit* est égal à *compteur* x *entier1* tout au long de l'algorithme.
Or à la fin de l'algorithme *compteur*=*entier2* donc l'algorithme est correct.