Introduction à l'algorithmique

et à la programmation structurée
et orientée objet
en Python

Christophe Rey

christophe.rey@univ-bpclermont.fr
Bureau 123 - 04.70.30.43.92

Département SRC de Vichy
IUT de Montluçon
Université Blaise Pascal Clermont 2

Plan du cours 1/4

I. Introduction
II. Etude de cas : la galette des rois
  A. Les recettes
  B. Représentation graphique : les organigrammes
    1. Agencement général des tâches
    2. Organigrammes peu ou très détaillés
  C. Des organigrammes au programme Python
    1. Codes Python
    2. Structure du code (entête de fonction, corps, instructions, commentaires)
    3. Autre exemple
    4. à 6. Toutes les autres fonctions
  D. La programmation structurée via l'appel de fonction
    1. Le principe de la programmation structurée et de l'appel de fonction
    2. Architecture
    3. Le code de la fonction principale
    4. Le code complet
    5. La traduction du code en un programme exécutable en Python

Plan du cours 2/4

III. Introduction à l'algorithmique
  A. Les instructions
    1. L'affectation
    2. L'instruction de calcul
    3. Les appels de fonctions
    4. Les entrées-sorties
  B. Les tests
    1. Les conditions
    2. Le test simple
    3. La fin des tests
    4. Tests successifs et imbriqués, liste de choix
    5. Conditions composées
  C. Les boucles
    1. 3 types de boucles, comment les choisir
    2. La sortie des boucles
    3. Liens entre les boucles
    4. Boucles successives et imbriquées
    5. Tant que avec plusieurs conditions d'arrêt
    6. Erreurs à ne pas faire
  D. Jamais de Goto
  E. Vérifier un organigramme

Plan du cours 3/4

IV. Introduction à la programmation structurée en Python
  A. Gestion de la mémoire
    1. Ruban mémoire, bit et octet
    2. Variable : définition et représentation
    3. Types
    4. Classes, références et objets
    5. Création de variables
    6. Manipulation de variables
  B. Fonctions
    1. Principe et exemples
    2. Syntaxe
    3. Appel de fonction: principe, paramètres, passage de paramètres par valeur ou référence, valeur de retour

Plan du cours 4/4

V.Introduction à la programmation orientée objet en Python

A.Notions fondamentales

1.Principe, définition
2.Utilisation d'un objet
3.Encapsulation, notion de constructeur et d'accesseur

B..Exercice

I. Introduction

Algorithme (cf Wikipédia):

Méthode de résolution de problème énoncée sous la forme d'une série d'opérations à effectuer.
Exemples : *recettes de cuisine, plan de montage, ...*

Implémentation :

Mise en œuvre de l'algorithme qui consiste en l'écriture des opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique.

Code source :

Texte écrit dans un langage de programmation et qui détaille l'enchaînement des opérations constituant un programme.

Bibliographie Python

- Apprendre à programmer avec Python - IREM des Pays de la Loire

www.irem.sciences.univ-nantes.fr/Calcul/python_notes.pdf

- http://python.org/

Site web officiel

II. Etude de cas :

La recette de la galette des rois

II.A. Les recettes

tirées de www.meilleurduchef.com

II.A.1. Recette de la galette des rois

Ingrédients (pour 8 personnes) :
500g de pâte feuilletée
250g de crème frangipane
1 jaune d'oeuf
1 fève

Récipients :
plan de travail
plaque de cuisson
grille

Outils :
rouleau à pâtisserie
pinceau
couteau
four
palette

1. Avec le rouleau , étaler la moitié de la pâte feuilletée .
2. Étaler au centre une couche de crème frangipane de 1 à 2 cm d'épaisseur en forme de cercle .
3. Avec le pinceau, dorer au jaune d 'œuf tout autour.
4. Si vous voulez y déposer une fève , c'est le moment ou jamais. Piquer votre fève dans la crème frangipane .
5. Après avoir étalé un second morceau de pâte feuilletée , le poser sur le premier morceau .
6. Appuyez sur le pourtour de la crème frangipane pour bien faire adhérer les deux morceaux de pâte .
7. Avec le couteau, découper votre galette en forme de cercle .
8. Chiqueter la bordure de votre galette .
9. Dorer toute la surface au jaune d 'œuf. "Vous pouvez dessiner avec le dos d 'un couteau d 'office en traçant des courbes ou des rainures ".
10. Enfourner à four chaud à 180-200°C. La pâte feuilletée va commencer à cuire et à gonfler . Sa surface va également commencer à dorer .
11. La galette est cuite lorsque le dessous de la pâte sera cuit . Pour vérifier sa cuisson, s'aider d 'une palette assez large et soulever le bord de la galette .
12. Quand la galette est cuite , la sortir du four et la laisser refroidir sur une grille . Elle se mange froide ou chaude. "Je vous conseille de la faire tiédir , elle n'en sera que meilleure ."

Source :http://www.meilleurduchef.com

II.A.2. Recette de la pâte feuilletée

Ingrédients (pour 500 g de pâte) :
250 g de farine
125g d'eau
1pincée de sel
185g de beurre

Récipients :
plan de travail

Outils :
rouleau à pâtisserie
couteau
papier film

Détrempe :
1. Disposer la farine en fontaine sur le plan de travail .
2. Y ajouter au centre , l'eau et le sel.
3. Commencer à malaxer du bout des doigts le centre de la fontaine en mélangeant petit à petit la farine et l'eau. Former une boule homogène .
4. Tailler la surface en croix avec un couteau .
5. Envelopper dans du papier film .
6. Laisser poser au frais 30 minutes minimum.

Tourage :
7. Étaler la détrempe en croix en se servant des incisions comme repaire . Laisser le centre un peu plus épais.
8. Disposer au centre , le beurre ramolli .
9. Rabattre les quatre cotés afin de l 'envelopper .
10. Égaliser l'épaisseur du pâton en tapotant dans les deux sens avec le rouleau à pâtisserie . La forme doit être carrée.

Faire 6 fois de suite les 3 opérations suivantes qui constituent ce qu'on appelle un « tour »:
11. Étaler en longueur pour obtenir une épaisseur de 1 cm et une longueur 3 fois plus longue que la largeur.
12. Plier cette bande en 3.
13. Faire pivoter la bande d '1/4 de tour.
14. Tous les 2 tours, laisser poser au frais 30 minutes minimum.

A la fin des 6 tours, la pâte est prête à être utilisée.

Source :http://www.meilleurduchef.com

II.A.3. Recette de la crème frangipane

Ingrédients (pour 1 galette de 8 personnes) :
100 g de beurre pommade ,
125 g sucre glace ,
125 g amandes en poudre ,
12 g de maïzena ,
12 g de rhum ,
75 g d'œufs,
225 g de crème pâtissière

Outils :
fouet

Récipients :
plan de travail
bassine de pâtisserie

1. Confectionner une crème pâtissière .
2. Dans la bassine, mélanger le sucre glace et les amandes en poudre : on obtient ce que l'on appelle un tant pour tant .
3. Blanchir le tant pour tant avec le beurre .
4. Ajouter les œufs, la maïzena et le rhum .
5. Mélanger la préparation à la crème pâtissière . La crème frangipane est prête à être utilisée.

Source :http://www.meilleurduchef.com

II.A.4. Recette de la crème pâtissière

Ingrédients (pour 1 litre) :
150g de sucre
75g de farine
5 jaunes d'oeufs
½ gousse de vanille
¾ de litre de lait

Récipients :
plan de travail
bassine de pâtisserie
casserole en inox
bol

Outils :
fouet
couteau
papier film

1. Blanchir les jaunes d'œuf avec le sucre dans la bassine.
2. Incorporer la farine .
3. Couper la gousse de vanille en deux dans le sens de la longueur avec le couteau.
4. Mettre le lait à bouillir avec la gousse de vanille dans la casserole .
5. Dès l'ébullition, verser en une seule fois le lait sur les œufs blanchis.
6. Fouetter à l'aide du fouet.
7. Transvaser le tout dans la casserole du lait .
8. Mettre à cuire sur le feu et, pendant 4 à 5 minutes, remuer constamment.
9. Débarrasser la crème dans un bol .
10. Couvrir d'un papier film.

Source :http://www.meilleurduchef.com

II.A.5. Comment blanchir un mélange

Ingrédients :
œufs (ou jaunes d'œufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine de pâtisserie

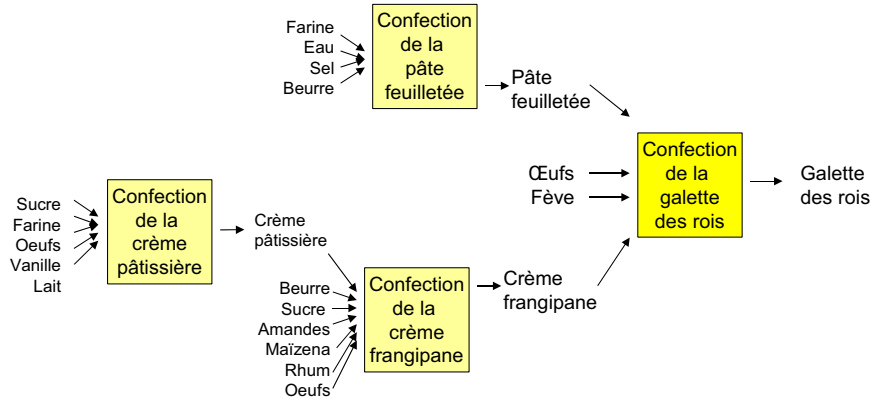
1. Dans la bassine, mélanger le sucre et les œufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.

Source :http://www.meilleurduchef.com

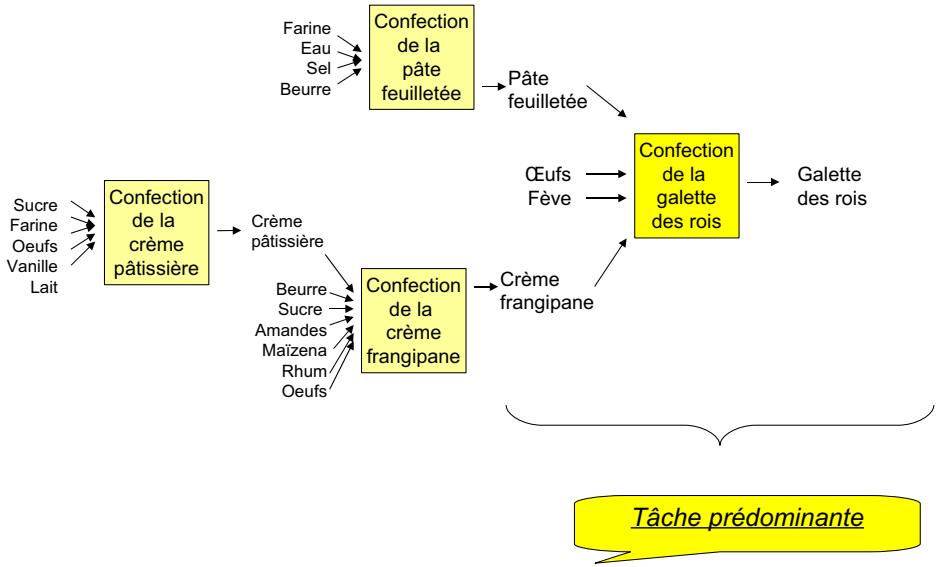
II.B. Les Organigrammes

Ou comment dessiner
ce qu'il y a à faire
de manière claire

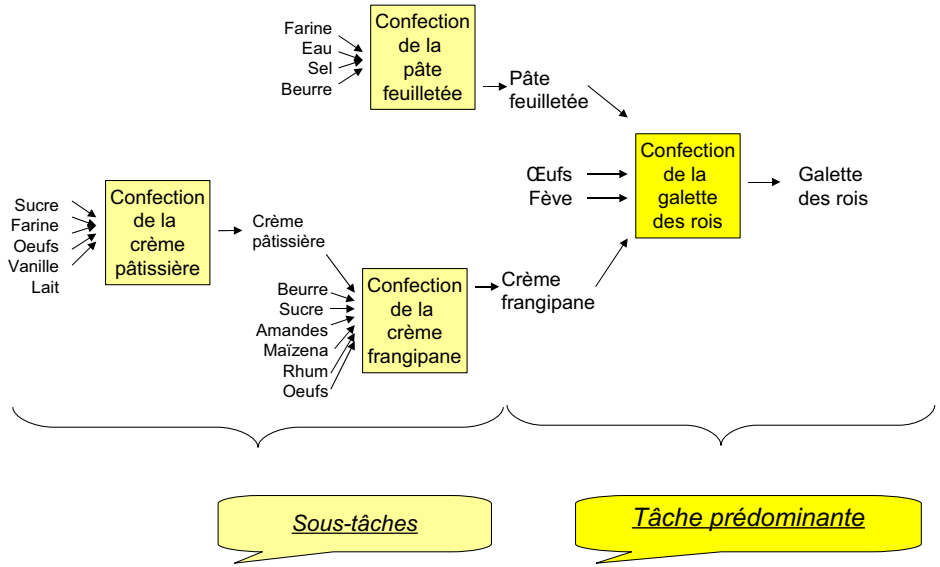
II.B.1. Agencement général des tâches



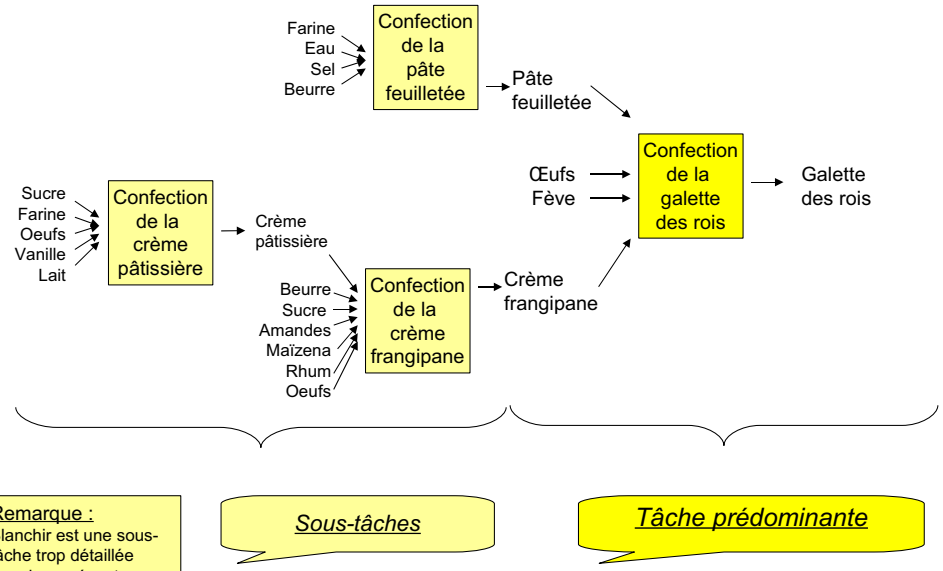
II.B.1. Agencement général des tâches



II.B.1. Agencement général des tâches

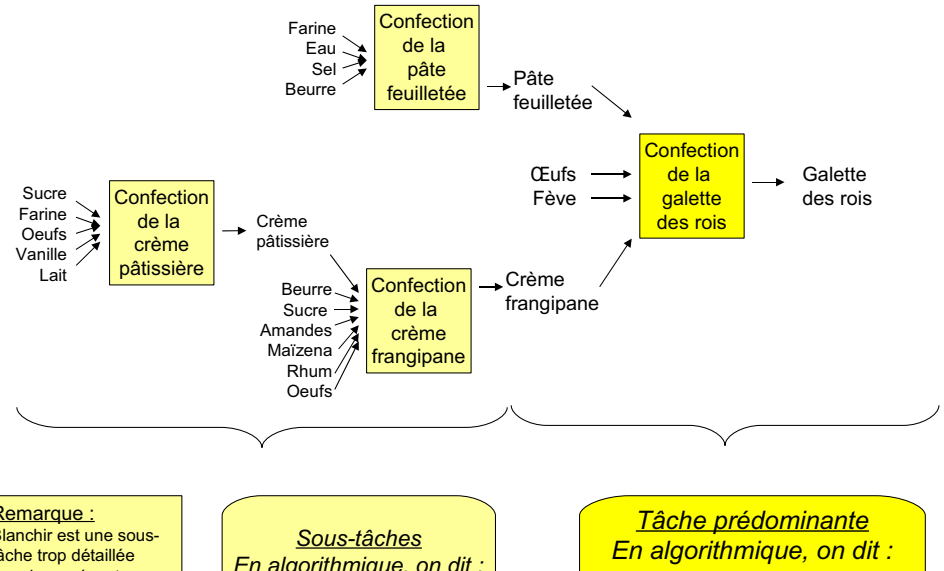


II.B.1. Agencement général des tâches



Remarque :
Blanchir est une sous-tâche trop détaillée pour la représenter dans l'agencement général.

II.B.1. Agencement général des tâches

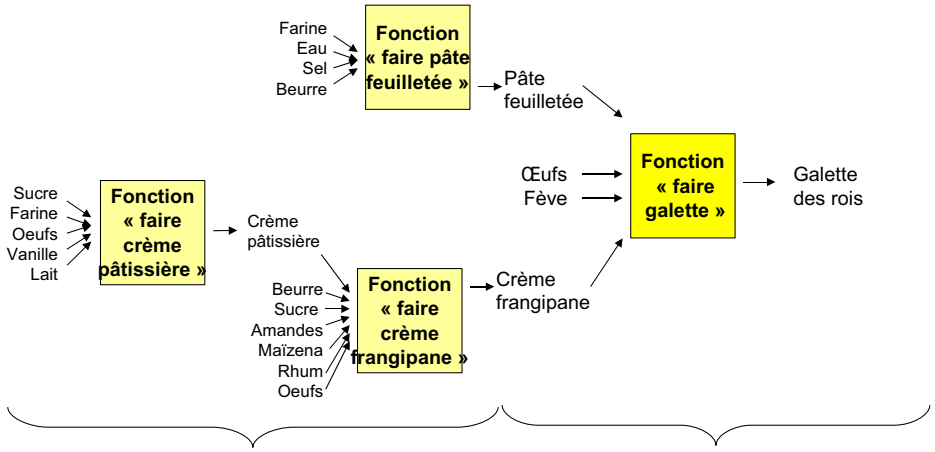


Remarque :
Blanchir est une sous-tâche trop détaillée pour la représenter dans l'agencement général.

Sous-tâches
En algorithmique, on dit :
*« **FONCTION** »*

Tâche prédominante
En algorithmique, on dit :
*« **FONCTION PRINCIPALE** »*

II.B.1. Agencement général des tâches

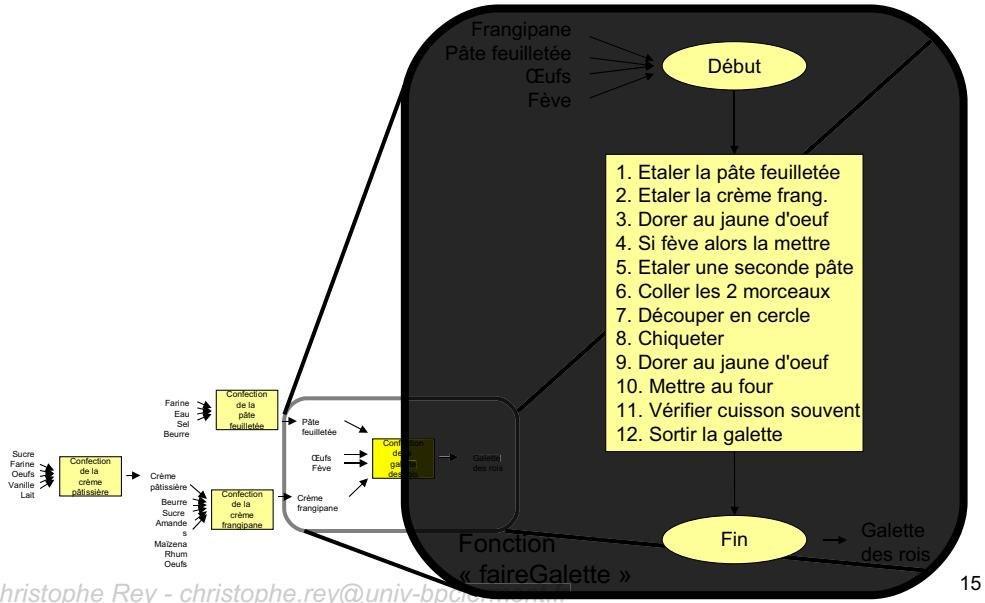


Remarque :
Blanchir est une sous-tâche trop détaillée pour la représenter dans l'agencement général.

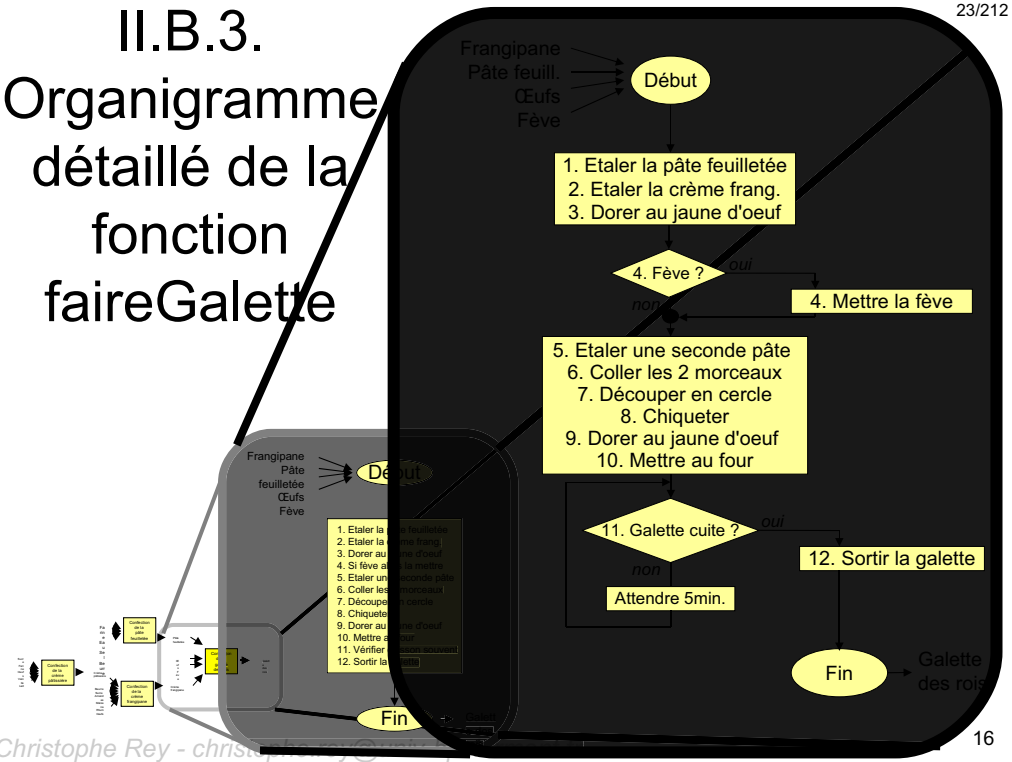
Sous-tâches
En algorithmique, on dit :
*« **FONCTION** »*

Tâche prédominante
En algorithmique, on dit :
*« **FONCTION PRINCIPALE** »*

II.B.2. Organigramme simple de la fonction « faireGalette »

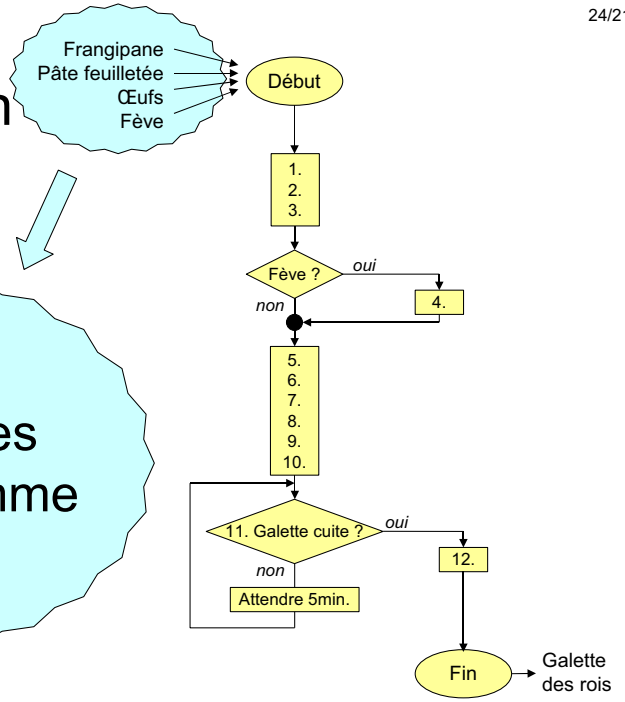


II.B.3. Organigramme détaillé de la fonction faireGalette

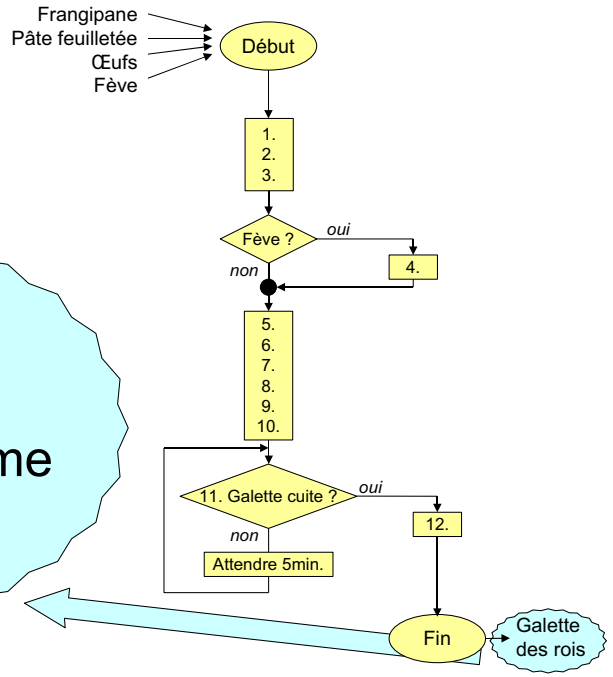


II.B.3. Fonction faireGalette

les entrées du programme

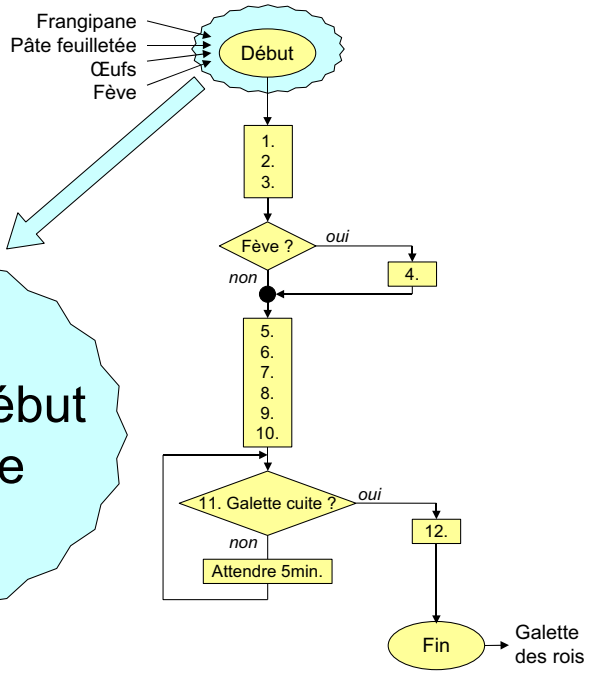


II.B.3. Fonction faireGalette



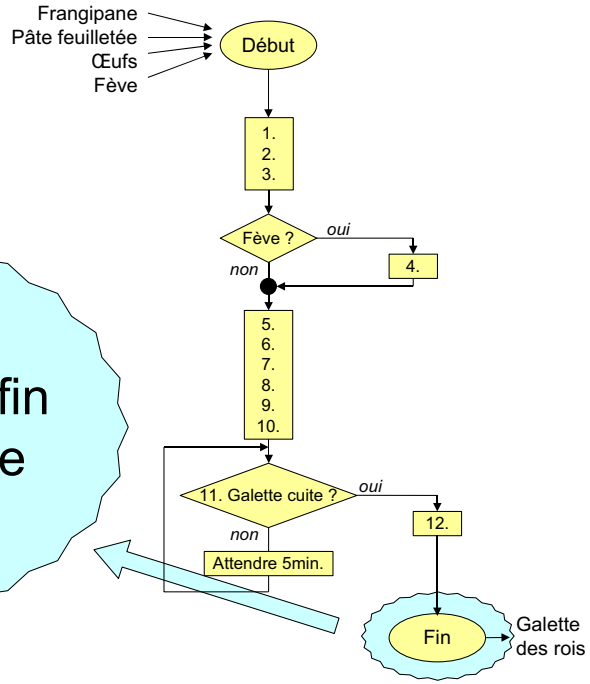
les sorties du programme

II.B.3. Fonction faireGalette



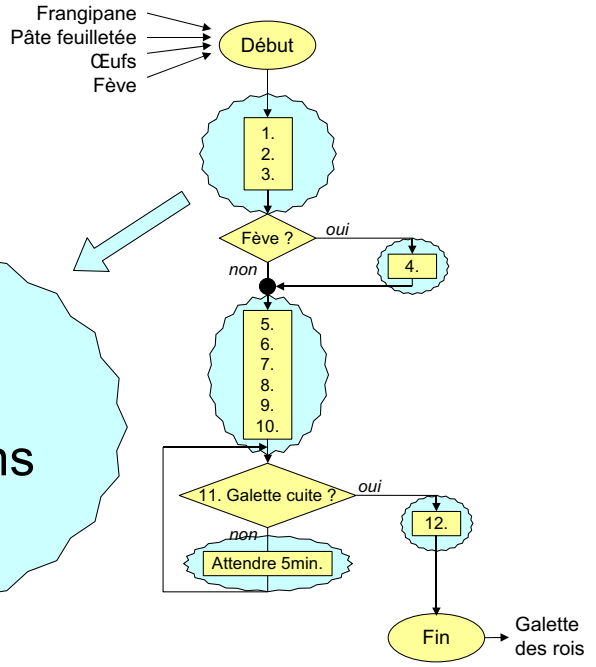
la balise de début d'algorithme

II.B.3. Fonction faireGalette



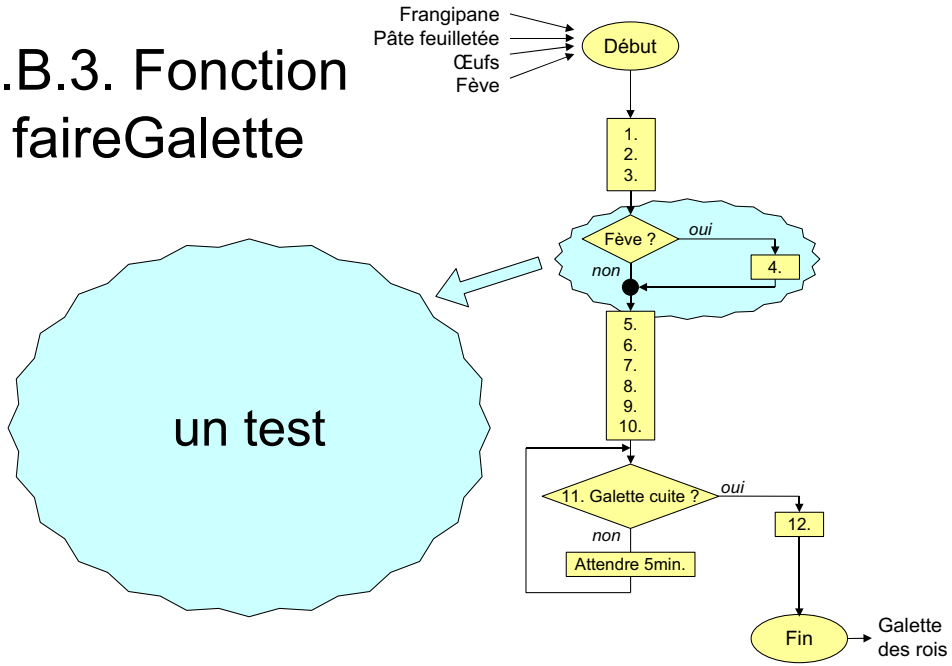
la balise de fin d'algorithme

II.B.3. Fonction faireGalette

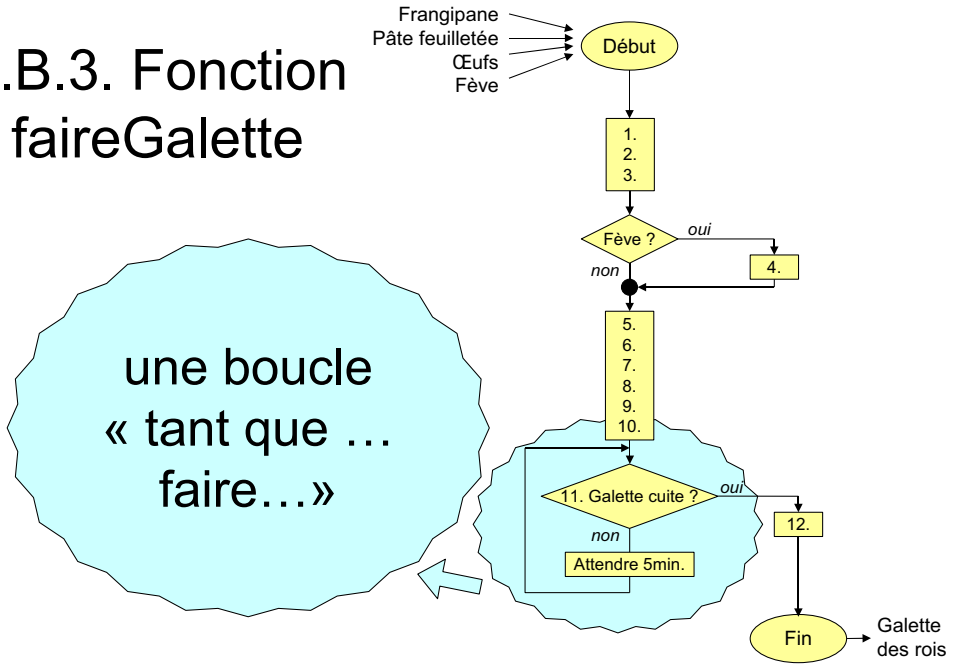


blocs d'instructions

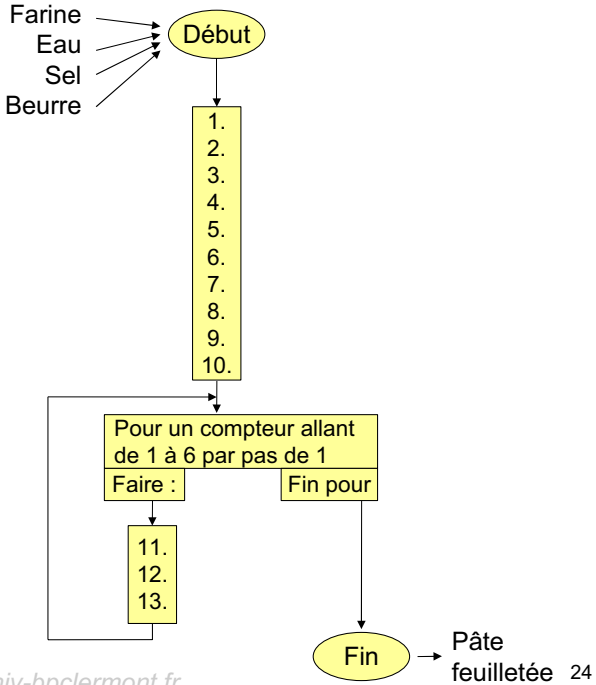
II.B.3. Fonction faireGalette



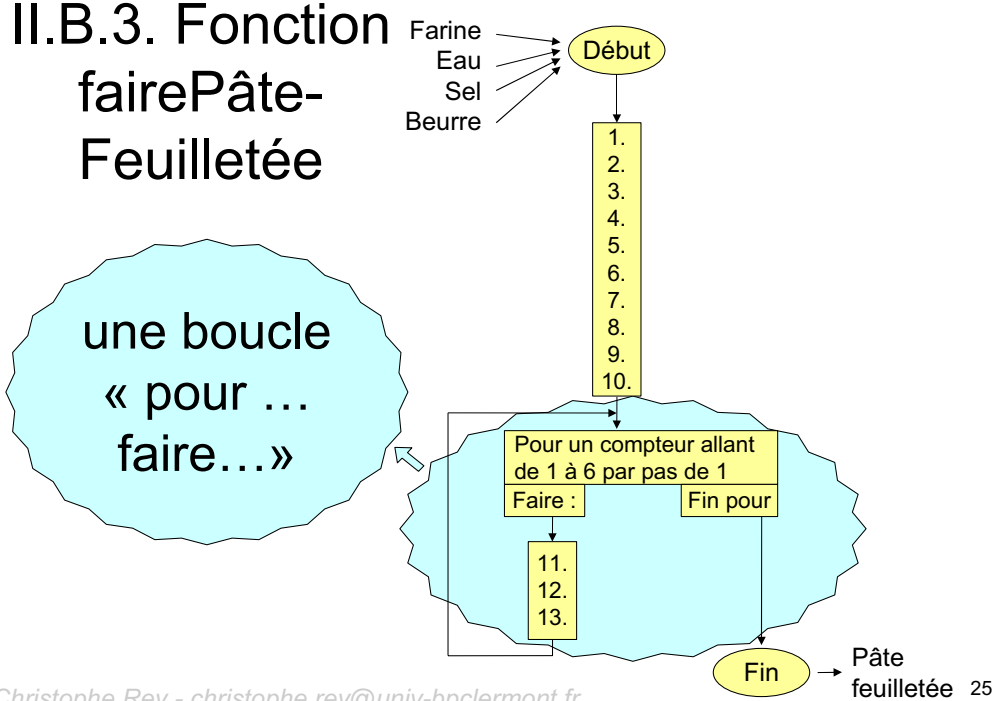
II.B.3. Fonction faireGalette



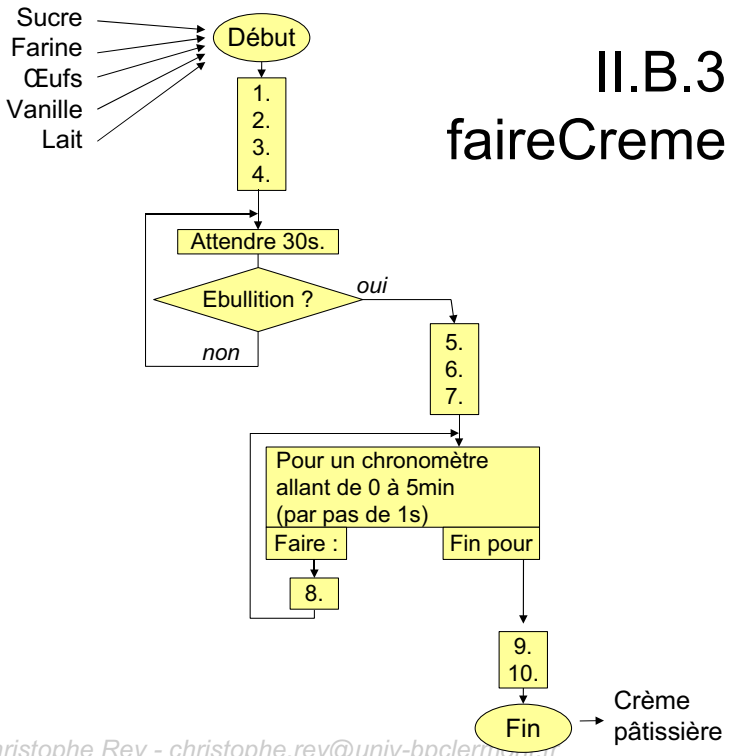
II.B.3. Fonction fairePâte-Feuilletée



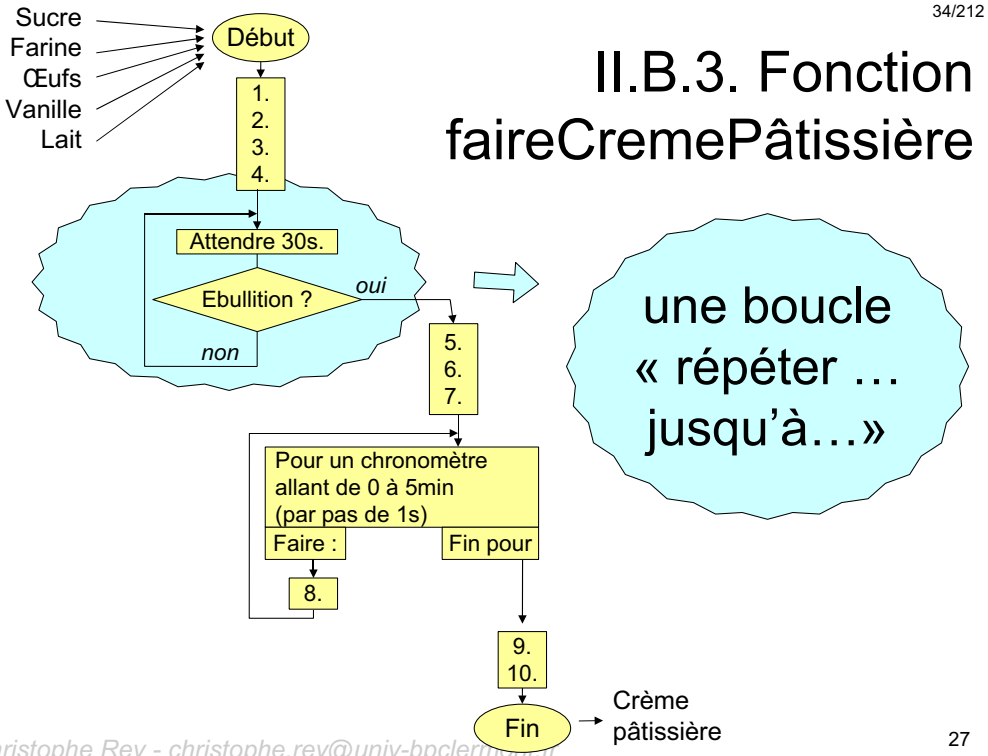
II.B.3. Fonction fairePâte-Feuilletée



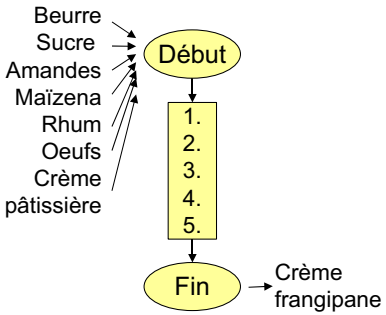
II.B.3. Fonction faireCremePâtissière



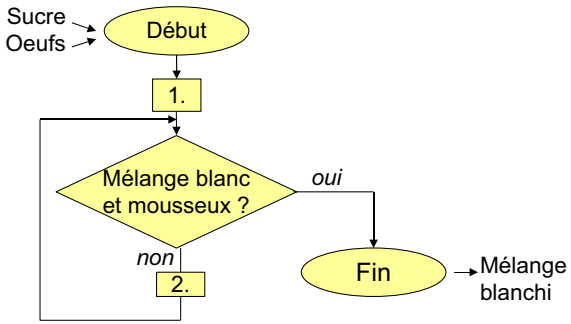
II.B.3. Fonction faireCremePâtissière



II.B.3 Fonction faireCrèmeFrangipane



II.B.3. Fonction blanchir



II.C. Codes sources façon Python Traduction des organigrammes

A quoi ressemblerait les fonctions précédentes si on les donnait à une machine pour que celle-ci fasse la recette à notre place ?

Remarque :
ce n'est pas grave si vous ne comprenez pas en détail maintenant, l'important est de comprendre le principe...

II.C.1. Code Python pour blanchir

```
############################################################
# Définition d'une fonction qu'on appelle "blanchir()".
# Cette fonction prend en entrées des oeufs (entrée nommée "o")
# et du sucre (entrée nommée "s").
# Cette fonction renvoie un mélange blanchi d'oeufs et de sucre.
# o (nombre d'oeufs)
# s (quantite de sucre)
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.

def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
                        # mélange des œufs et du sucre donnés
                        # en entrées.
    while m non blanchi:      # Tant que le mélange n'a pas blanchi,
        fouetter(m)     # je fouette ce mélange. Pour cela,
                        # je fais appel à la fonction fouetter().
    return m            # Je renvoie le mélange m blanchi
                        # en résultat de la fonction blanchir().
```


II.C.1. Code Python pour blanchir

Un code source est contenu dans un fichier texte.

Ici, on a le code source correspondant à la fonction « blanchir un mélange ».

II.C.2. Structure d'un code Python

Zone de commentaires initiale

II.C.2. Structure d'un code Python

Zone du code source de la fonction

```
def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
                        # mélange des œufs et du sucre donnés
                        # en entrées.
    while m non blanchi:      # Tant que le mélange n'a pas blanchi,
        fouetter(m)     # je fouette ce mélange. Pour cela,
                        # je fais appel à la fonction fouetter().
    return m            # Je renvoie le mélange m blanchi
                        # en résultat de la fonction blanchir().
```

II.C.2. Structure d'un code Python

```
############################################################
# Définition d'une fonction qu'on appelle "blanchir()".
# Cette fonction prend en entrées des oeufs (entrée nommée "o")
# et du sucre (entrée nommée "s").
# Cette fonction renvoie un mélange blanchi d'oeufs et de sucre
# o (nombre d'oeufs)
# s (quantite de sucre)
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.

def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
                        # mélange des œufs et du sucre donnés
                        # en entrées.
    while m non blanchi:      # Tant que le mélange n'a pas blanchi,
        fouetter(m)     # je fouette ce mélange. Pour cela,
                        # je fais appel à la fonction fouetter().
    return m            # Je renvoie le mélange m blanchi
                        # en résultat de la fonction blanchir().
```

En-tête de la fonction

Décrit :
- le nom de la fonction
- les entrées

II.C.2. Structure d'un code Python

```
############################################################
# Définition d'une fonction qu'on appelle "blanchir()".
# Cette fonction prend en entrées des oeufs (entrée nommée "o")
# et du sucre (entrée nommée "s").
# Cette fonction renvoie un mélange blanchi d'oeufs et de sucre
# o (nombre d'oeufs)
# s (quantite de sucre)
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.

def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
                        # mélange des œufs et du sucre donnés
                        # en entrées.
    while m non blanchi:      # Tant que le mélange n'a pas blanchi,
        fouetter(m)     # je fouette ce mélange. Pour cela,
                        # je fais appel à la fonction fouetter().
    return m            # Je renvoie le mélange m blanchi
                        # en résultat de la fonction blanchir().
```

Corps de la fonction

II.C.2. Structure d'un code Python

```
############################################################
# Définition d'une fonction qu'on appelle "blanchir()".
# Cette fonction prend en entrées des oeufs (entrée nommée "o")
# et du sucre (entrée nommée "s").
# Cette fonction renvoie un mélange blanchi d'oeufs et de sucre
# o (nombre d'oeufs)
# s (quantite de sucre)
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.

def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
                        # mélange des œufs et du sucre donnés
                        # en entrées.
    while m non blanchi:      # Tant que le mélange n'a pas blanchi,
        fouetter(m)     # je fouette ce mélange. Pour cela,
                        # je fais appel à la fonction fouetter().
    return m            # Je renvoie le mélange m blanchi
                        # en résultat de la fonction blanchir().
```

Instructions de la fonction

II.C.2. Structure d'un code Python

```
############################################################
# Définition d'une fonction qu'on appelle "blanchir()".
# Cette fonction prend en entrées des oeufs (entrée nommée "o")
# et du sucre (entrée nommée "s").
# Cette fonction renvoie un mélange blanchi d'oeufs et de sucre
# o (nombre d'oeufs)
# s (quantite de sucre)
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.

def blanchir(o, s):
    m = o + s;          # Je définis un mélange m comme le
```

Instructions de la fonction

Une instruction :
tâche la plus simple qu'on ne décompose pas en sous-tâches.
Ex. : mélanger, verser un ingrédient,...

II.C.2. Structure d'un code Python

Commentaires des instructions

```
##############################################################
# Définition d'une fonction                                  #
# Cette fonction prend en en                          "o")   #
# et du sucre (entrée nommée                                 #
# Cette fonction renvoie un                           sucre. #
# o (nombre d'oeufs)                                         #
# s (quantite de sucre)                                      #
# Le résultat retourné est un mélange d'oeufs et de sucre blanchi.
##############################################################

def blanchir(o, s):
    m = o + s;              # Je définis un mélange m comme le
                            # mélange des œufs et du sucre donnés
                            # en entrées.
    while m non blanchi:        # Tant que le mélange n'a pas blanchi,
        fouetter(m)         # je fouette ce mélange. Pour cela,
                            # je fais appel à la fonction fouetter().
    return m                # Je renvoie le mélange m blanchi
                            # en résultat de la fonction blanchir().
```

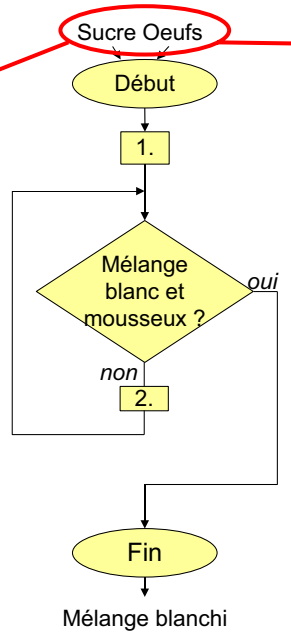

II.C.2. Obtention d'un code Python

Ingrédients :
oeufs (ou jaunes d'oeufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine à pâtisserie

Recette :
1. Dans la bassine, mélanger le sucre et les oeufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.




```
def blanchir(o, s):
    m = o + s;
    while m non blanchi:
        fouetter(m)
    return m
```

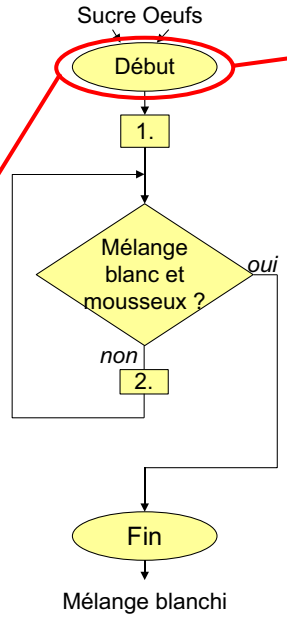

II.C.2. Obtention d'un code Python

Ingrédients :
oeufs (ou jaunes d'oeufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine à pâtisserie

Recette :
1. Dans la bassine, mélanger le sucre et les oeufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.




```
def blanchir(o, s):
    m = o + s;
    while m non blanchi:
        fouetter(m)
    return m
```

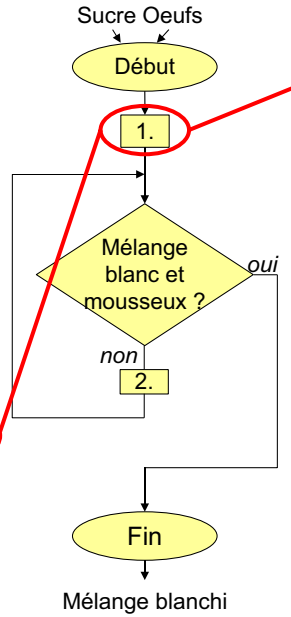

II.C.2. Obtention d'un code Python

Ingrédients :
oeufs (ou jaunes d'oeufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine à pâtisserie

Recette :
1. Dans la bassine, mélanger le sucre et les oeufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.




```
def blanchir(o, s):
    m = o + s;
    while m non blanchi:
        fouetter(m)
    return m
```

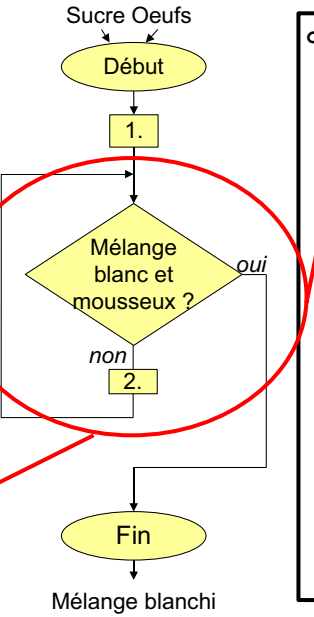
II.C.2. Obtention d'un code Python

Ingrédients :
oeufs (ou jaunes d'oeufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine à pâtisserie

Recette :
1. Dans la bassine, mélanger le sucre et les oeufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.




```
def blanchir(o, s):
    m = o + s;
    while m non blanchi:
        fouetter(m)
    return m
```

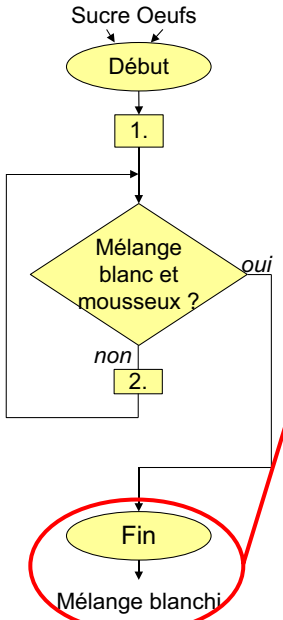

II.C.2. Obtention d'un code Python

Ingrédients :
oeufs (ou jaunes d'oeufs)
sucre

Outils :
fouet

Récipients :
plan de travail
bassine à pâtisserie

Recette :
1. Dans la bassine, mélanger le sucre et les oeufs.
2. Tant que le mélange n'est ni blanc ni mousseux, le battre au fouet.




```
def blanchir(o, s):
    m = o + s;
    while m non blanchi:
        fouetter(m)
    return m
```


II.C.3, 4, 5. Autres fonctions

```
def fairePateFeuilletee(farine, eau, sel, beurre):
    ... # Instructions de la fonction
```

```
def faireCremeFrangipane
(beurre, oeufs, sucre, amande, maizena, rhum rh,
cremePatissiere):
    ... # Instructions de la fonction
```

```
def fairePateFeuilletee(farine, eau, sel, beurre):
    ... # Instructions de la fonction
```


II.C.6 Code fonction principale

- On a traduit les fonctions :
  - Blanchir
  - FaireCrème patissière
  - FaireCrèmeFrangipane
  - FairePâteFeuilletée
- Il reste la fonction principale FaireGalette
  - En Python, c'est le code qui suit les définitions des fonctions.
  - En interne, c'est la fonction « __main__ »
  - La fonction principale est celle qui « gère » toutes les autres : on dit qu'**elle les appelle.**

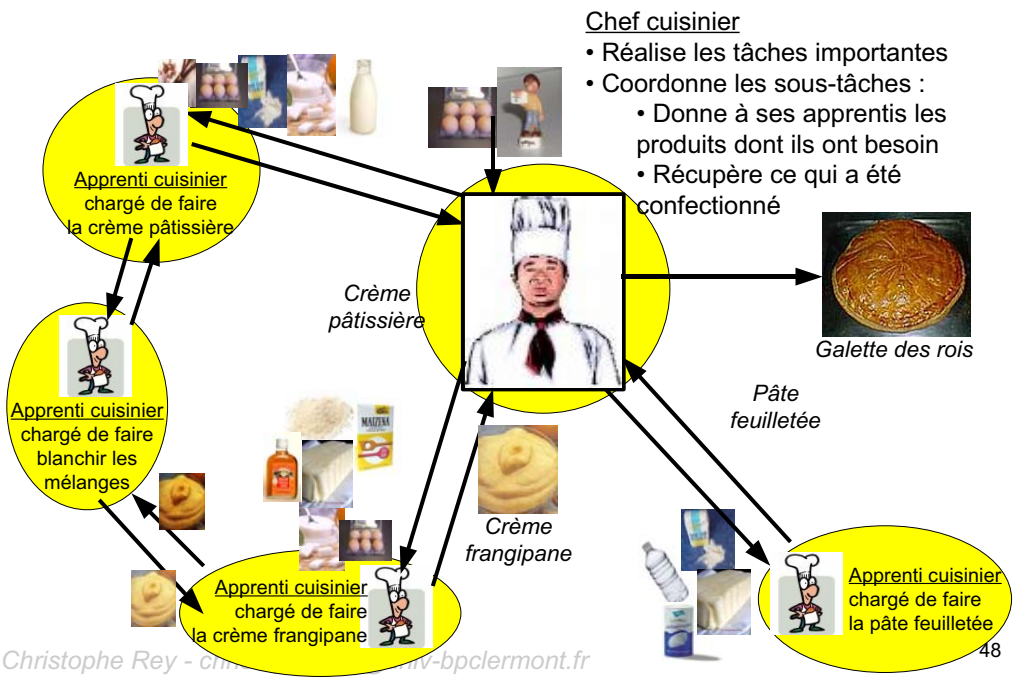
II.D. La programmation structurée via l'appel de fonction

Certaines fonctions en appellent d'autres.

II.D.1 La programmation structurée

- Principe (rappel) :
  - Décomposer le problème en sous-problèmes (sous-tâches) plus simples
  - Chaque sous-problème correspond à une fonction
  - Les fonctions communiquent grâce aux appels de fonctions.
- Principe de l'appel de fonction
  Une fonction f1 appelle une fonction f2 quand :
  - f1 est en cours d'exécution
  - f1 a besoin du résultat de l'exécution de f2
  - f1 ordonne à f2 de s'exécuter
  - f2 s'exécute et termine et alors f1 peut récupérer son résultat et s'en servir

II.D.1. La prog. structurée, c'est :



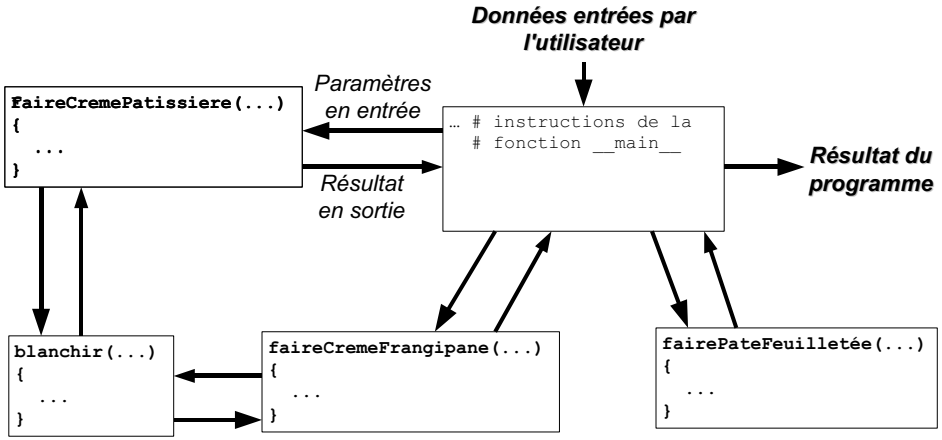
II.D.1. La prog. structurée, ce N'est :

NI un cuisinier SEUL qui réalise toutes les tâches.

```
...   # Instructions pour faire
...   # la crème pâtissière.

...   # Instructions pour faire
...   # la crème frangipane.

...   # Instructions pour faire
...   # la pâte feuilletée.

...   # Instructions pour faire
...   # la galette des rois.
```

NI une fonction __main__ SEULE qui contient toutes les instructions

II.D.2. Architecture d'un programme :



Remarque
Les fonctions autres que __main__ peuvent aussi s'appeler entre elles. Par exemple faireCremePatissiere() appelle blanchir().

La fonction __main__
- Réalise les tâches importantes
- Appelle les autres fonctions :
  - Leur transmet les données dont elles ont besoin (appelées paramètres)
  - Récupère le résultat de leur traitement

II.D.3 Code du programme

```
################################################
# Programme (fonction __main__)

# on fait la crème patissière
cp = faireCremePatissiere(5, 150, 75, 0.5, 750)

# on divise la quantité obtenue par 4 pour la galette
cp = cp / 4

# on fait la crème frangipane
cf = faireCremeFrangipane(100, 75, 125, 125, 12, 12, cp);

# on fait la pâte feuilletée
pf = fairePateFeuilletee(250, 125, 1, 185);

... // Instructions pour faire la galette des rois à partir
... // de la pâte et des crèmes préparées précédemment
```


II.D.3 Code du programme

```
################################################
# Programme (fonction __main__)

# on fait la crème patissière
cp = faireCremePatissiere(5, 150, 75, 0.5, 750)

# on divise la quantité obtenue par 4 pour la galette
cp = cp / 4

# on fait la crème frangipane
cf = faireCremeFrangipane(100, 75, 125, 125, 12, 12, cp);

# on fait la pâte feuilletée
pf = fairePateFeuilletee(250, 125, 1, 185);

... // Instructions pour faire la galette des rois à partir
... // de la pâte et des crèmes préparées précédemment
```

3 appels de fonctions dans la fonction main

II.D.4. Le code source complet

```
################################################
# Programme de fabrication d'une galette des rois
# auteur : Christophe Rey, Vichy, 2011
################################################

################################################
# Importation de fonctions externes

from cuisine import *

################################################
# Définitions locales de fonctions

def blanchir(o, s):
... # Instructions de la fonction

def fairePateFeuilletee(farine, eau, sel, beurre):
... # Instructions de la fonction

def faireCremeFrangipane (beurre, oeufs, sucre, amande, maizena, rhum rh,
cremePatissiere):
... # Instructions de la fonction

def fairePateFeuilletee(farine, eau, sel, beurre):
... # Instructions de la fonction

################################################
# Programme (fonction __main__)

... // Instructions pour faire la galette des rois à partir
... // de la pâte et des crèmes préparées précédemment
```

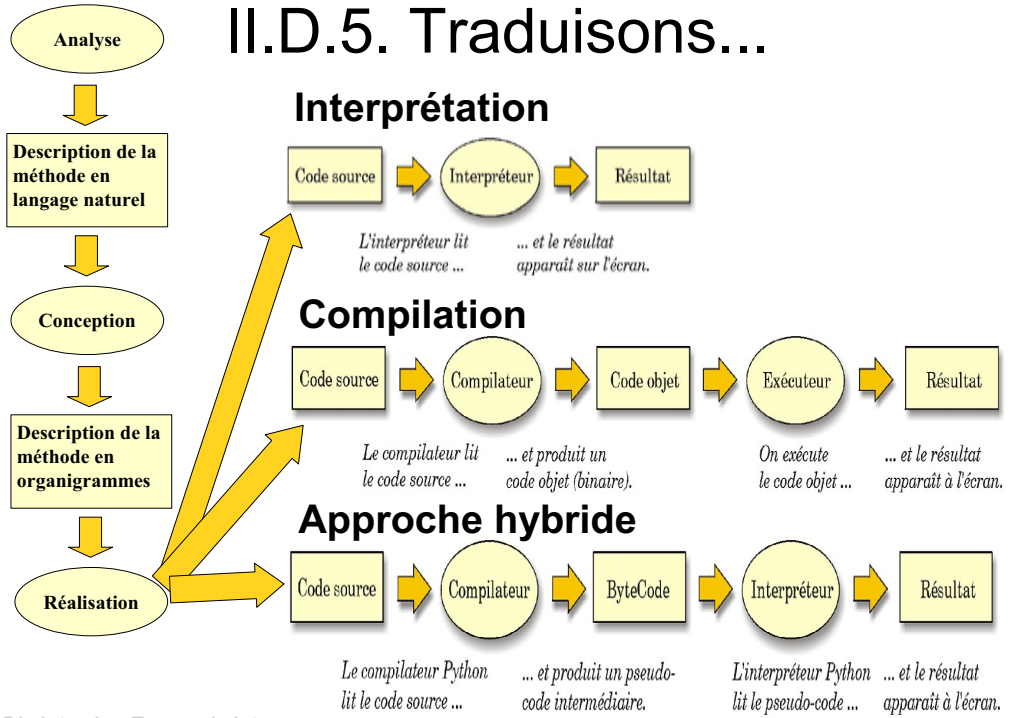
II.D.4. Le code source complet

```
################################################
# Programme de fabrication d'une galette des rois
# auteur : Christophe Rey, Vichy, 2011
################################################

################################################
# Importation de fonctions externes

from cuisine import *

##########
# Définitio

def blanchi
... # Instr

def fairePa
... # Instr

def faireCr
cremePatiss
... # Instr

def fairePa
... # Instr

################################################
# Programme (fonction __main__)

... // Instructions pour faire la galette des rois à partir
... // de la pâte et des crèmes préparées précédemment
```

ICI, on a un programme simple :
Un seul fichier texte contient les codes sources de toutes les fonctions.

EN GENERAL :
les codes sources sont répartie dans plusieurs fichiers texte.

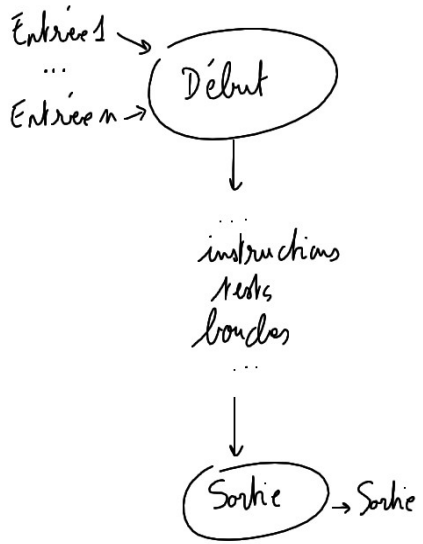
II.D.5. Traduisons...



III. Introduction à l'algorithmique

Rappels

- Un algorithme est
  - une suite
    - d'instructions,
    - de tests,
    - et de boucles,
  - qui peut avoir besoin de valeurs en entrée
  - et qui peut renvoyer un résultat en sortie.

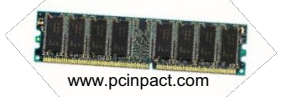


III.A. Les instructions

III.A. Métaphore culinaire filée

- Plan de travail = mémoire de l'ordinateur
- Récipients = variables (cases de la mémoire)
- Ingrédients = valeurs manipulées (entiers, réels, caractères,...)

III.A. Ruban mémoire



www.pcinpact.com

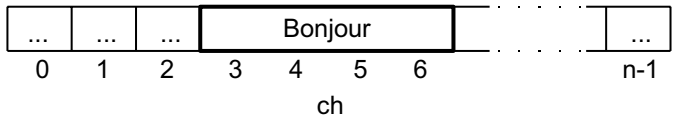
Une barette de mémoire

Le ruban mémoire

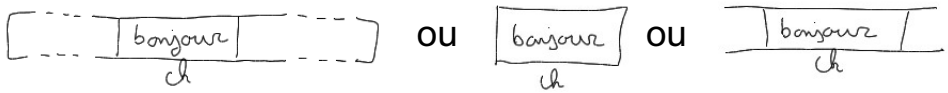
III.A. Ruban mémoire simplifié

- Le principe :
  - une case = une variable (sachant qu'une variable peut occuper plusieurs octets)
  - Les noms des variables sont notés en dessous
- Par exemple :

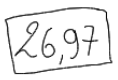
Une variable « ch » qui occupe 4 octets et qui stocke une chaîne de caractères.



On pourra aussi dessiner cette variable ainsi :



De même pour les variables numériques :



III.A. Instruction

- Action de base que la machine exécute
- Exemples
  - Ajouter un ingrédient dans un récipient / Stocker une valeur dans une variable (on dit affecter la valeur à la variable
  - Poser un récipient sur l'espace de travail / Créer une nouvelle variable en mémoire
  - Demander à un apprenti de faire une crème / Appeler une fonction pour faire un traitement
  - ... / ...

III.A. Représentation d'une instruction

	Représentation dans un organigramme	Syntaxe en Python
Une seule Instruction	Instruction 1	Instruction 1
Un bloc d'instructions	Instruction 1 Instruction 2 … Instruction n	Instruction 1; Instruction 2; … Instruction n;

III.A.1. L'affectation

- Affecter une valeur à une variable, c'est stocker la valeur dans les cases mémoire qu'occupe la variable

Représentation dans un organigramme	Syntaxe en Python
i ← 14	i = 14

- Ainsi après exécution de cette instruction, on aura la variable suivante en mémoire :

III.A.1. Variable : contenant ou contenu

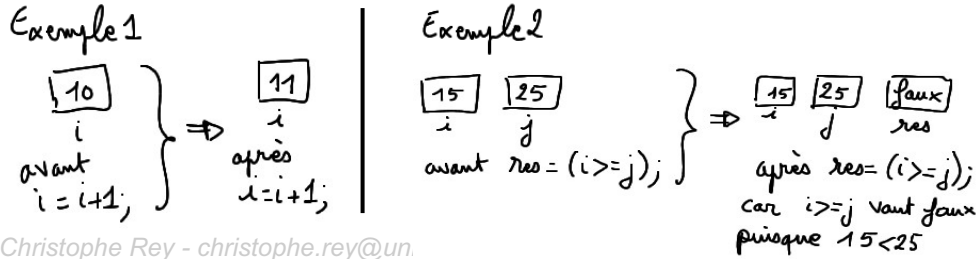
- Placée à gauche d'une affectation, une variable est considérée comme un contenant : on s'en sert pour stocker une valeur
- Placée ailleurs (par exemple à droite d'une affectation, ou dans un appel de fonction), une variable est considérée comme un contenu : on se sert de la valeur qu'elle contient

III.A.1. Variable : contenant ou contenu

- Ainsi il y a un abus de langage :
  - Tantôt variable est à comprendre comme récipient (la case de la mémoire)
  - Tantôt variable est à comprendre comme ingrédient (la valeur dans la case)

III.A.2. L'instruction de calcul

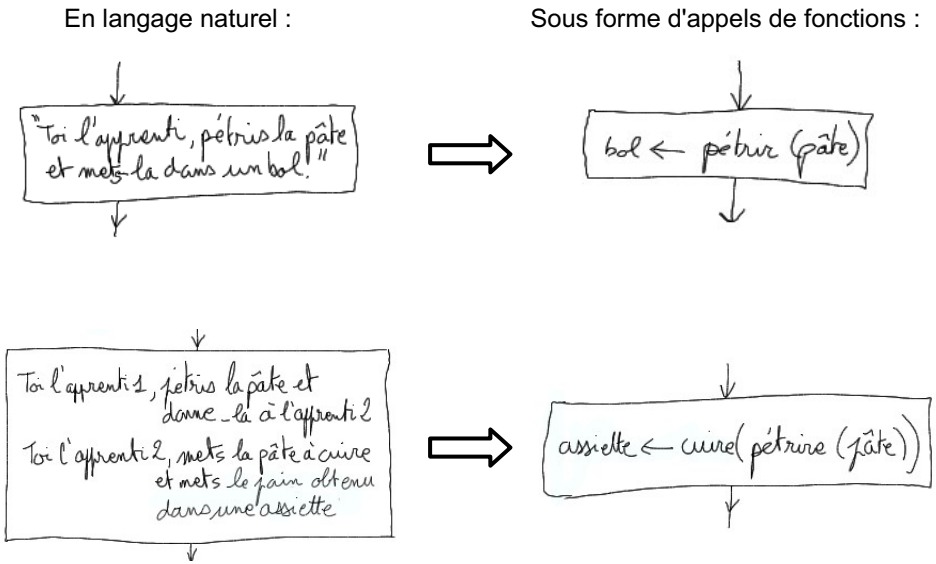
	Représentation dans un organigramme	Syntaxe en Python
Exemple 1	i ← i + 1	i = i + 1
Exemple 2	res ← (i >= j)	res = (i>=j)



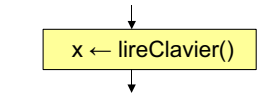
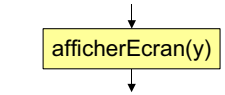
III.A.3. Les appels de fonctions

	Représentation dans un organigramme	Syntaxe en Python
Appel de fonction	i ← racineCarrée(57)	i = sqrt(57)
Appel de procédure	afficherEcran(i)	print(i)
Appel imbriqué	afficherEcran(racineCarrée(57))	print(sqrt(57))

III.A.3. Autres exemples



III.A.4. Les entrées/sorties au clavier/écran

	Représentation dans un organigramme	Syntaxe en Python
Lecture à partir du clavier		x=input() ou x=raw_input()
Affichage textuel à l'écran		print(y)

III.B. Les tests

III.B.1. A quoi cela sert-il ?

- Quand les traitements à faire peuvent varier en fonctions de certaines conditions.
- Exemples :
  - Si on veut mettre une fève dans la galette, alors il faut ajouter une instruction « mettre fève » dans la recette, sinon, on ne met rien.
  - Si on a du beurre qui n'est pas frais, alors on peut mettre du beurre demi-sel, mais dans ce cas il faut mettre moins de sel que prévu.
  - ...

III.B.1. Les conditions

- Dans une recette : on peut spécifier n'importe quelle condition exprimable en français.
- Dans un algorithme :
  - on travaille avec des nombres (et des chaînes de caractères) en mémoire
  - conditions sur ces nombres (et chaînes de caractères)
- Le résultat d'une condition est soit « true » (vrai, ou oui), soit « false » (faux, ou non).

III.B.1. Les conditions

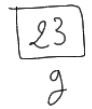
- Expressions contenant
  - Un opérateur de comparaison
  - Deux opérandes
- Opérateurs de comparaison :
  - "Inférieur à", noté <=
  - "Inférieur strictement à", noté <
  - "Supérieur à", noté >=
  - "Supérieur strictement à", noté >
  - "Égal à", noté ==
  - "Différent de", noté !=

III.B.1. Les conditions

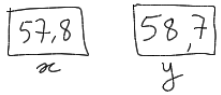
- Exemples

23 <= 45 est vrai
24 < 12 est faux
i >= 33 est faux si on a par exemple

-32 > -45 est vrai
g == 22 est faux si on a par exemple

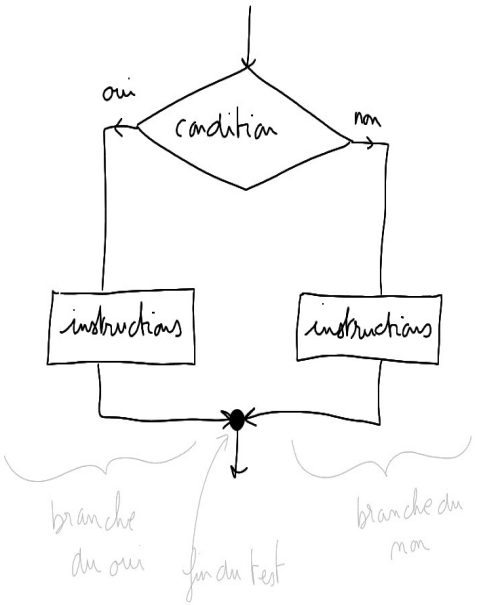


x != y est vrai si on a par exemple

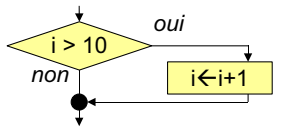
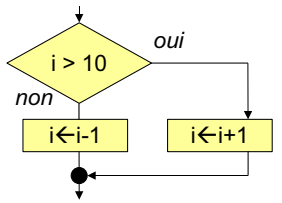
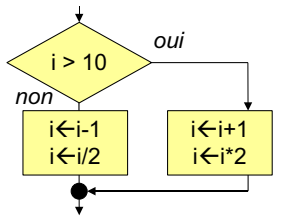


III.B.2. Le test

- Un test est symbolisé par un losange.
- Le losange contient la condition du test.
- Il divise le déroulement de l'algorithme en deux branches exclusives (la branche oui et la branche non).
- Il se termine toujours : avant la fin de l'algorithme, les 2 branches se rejoignent obligatoirement.
- Symbole de la fin des tests : un gros point noir.

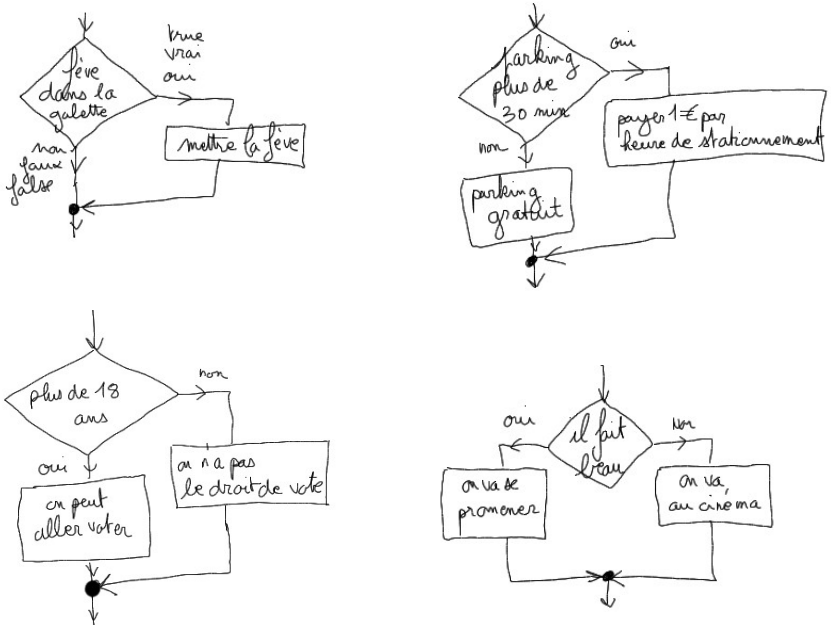


III.B.2. Le test

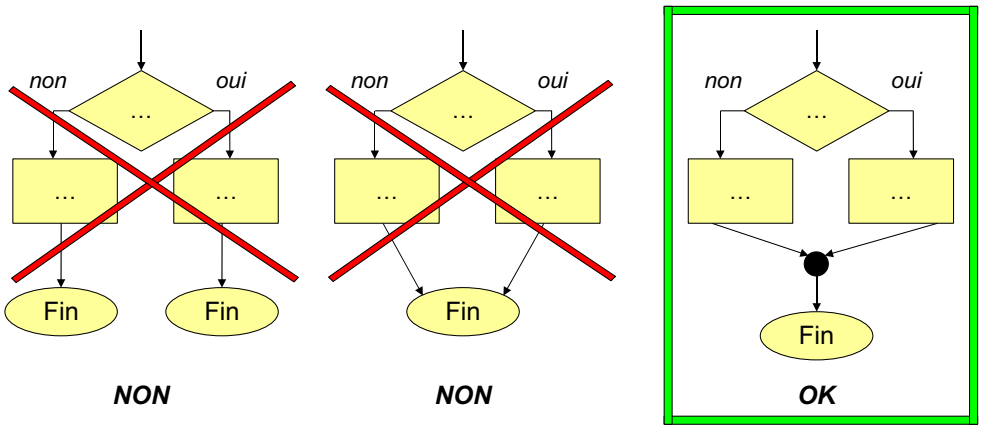
	Organigramme	Python
« **si** le contenu de la variable i est supérieur à 10 **alors** ajouter 1 à ce contenu, **fin_si** »		**if(i>10):** **i = i+1**
« **si** le contenu de la variable i est supérieur à 10 **alors** ajouter 1 à ce contenu, **sinon** oter 1 à ce contenu, **fin_si** »		**if(i>10):** **i = i+1** **else:** **i = i-1**
« **si** i est supérieur à 10 **alors** ajouter 1 à ce contenu puis le multiplier par 2, **sinon** oter 1 à ce contenu puis le diviser par 2 **fin_si** »		**if(i>10):** **i = i+1** **i = i*2** **else:** **i = i-1** **i = i/2**

Indentation obligatoire !!!!

III.B.2. Autres exemples de tests



III.B.3. La fin des tests



III.B.4.a. Tests successifs

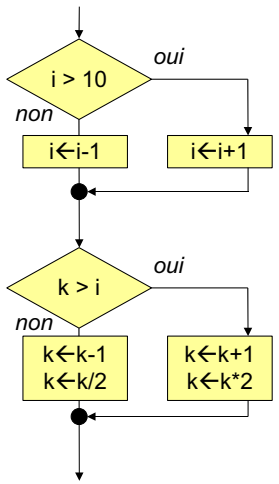
«

si le contenu de la variable i est supérieur à 10
alors
ajouter 1 à ce contenu,
sinon
oter 1 à ce contenu,
fin_si

si k est supérieur à i
alors
ajouter 1 à k
puis multiplier k par 2,
sinon
oter 1 à k
puis diviser k par 2
fin_si

»

Organigramme

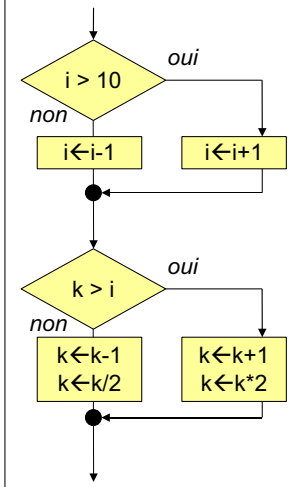


Python

```
if(i>10):
    i = i+1
else:
    i = i-1

if(k>i):
    k = k+1
    k = k*2
else:
    k = k-1
    k = k/2
```


III.B.4.a. Tests successifs

Organigramme



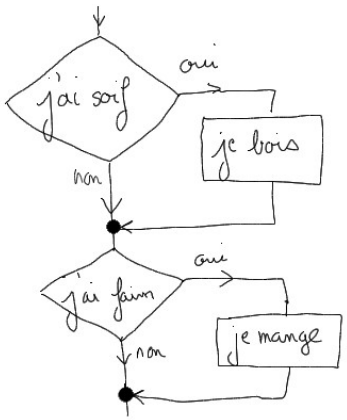
Python

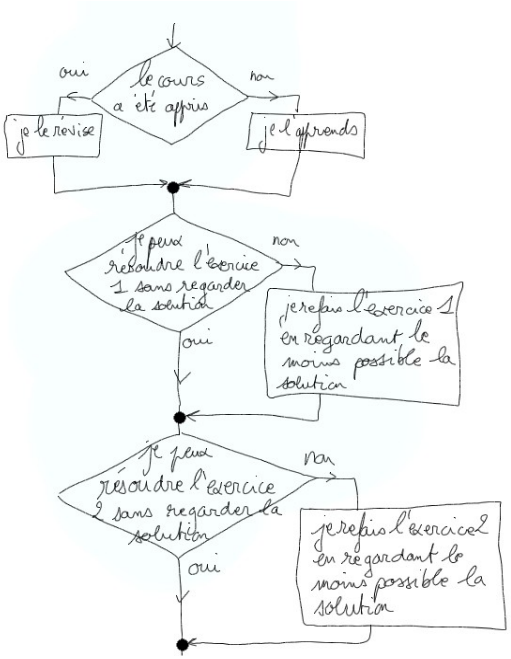
```
if(i>10):
    i = i+1
else:
    i = i-1

if(k>i):
    k = k+1
    k = k*2
else:
    k = k-1
    k = k/2
```

Exemple d'exécution :

- Départ : i = 12, k = 14
- Comme i > 10 alors on ajoute 1 à i. D'où : i = 13
- Comme k > i (puisque 14 > 13) alors on ajoute 1 à k : k = 15
  et on multiplie k par 2 en ajoutant le résultat dans k : k = 30

III.B.4.a. Tests successifs : autres ex.





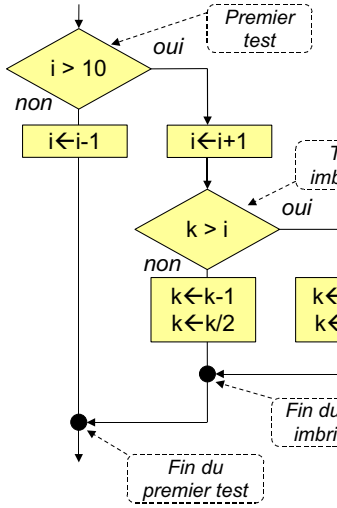
III.B.4.b. Tests imbriqués

«

si i est supérieur à 10
alors
  ajouter 1 à i
  si k est supérieur à i
  alors
    ajouter 1 à k
    puis multiplier k par 2,
  sinon
    oter 1 à k
    puis diviser k par 2
  fin_si
sinon
  oter 1 à i
fin_si

»

Organigramme

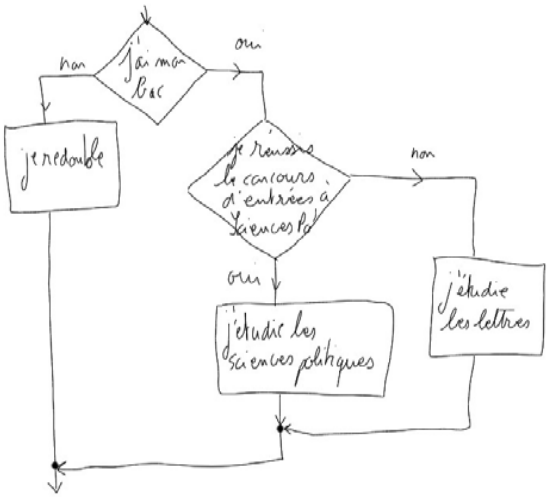


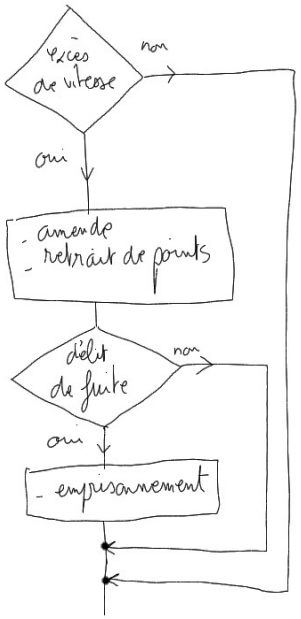
Python

```
if(i>10):
   i = i+1
   if(k>i):
      k = k+1
      k = k*2
   else:
      k = k-1
      k = k/2
else:
   i = i-1
```

Attention à l'indentation !!!!

III.B.4.b. Autres exemples





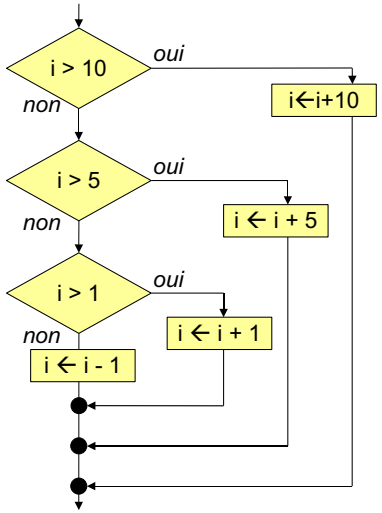
III.B.4.b. Tests imbriqués avec « else if »

«

si i est supérieur à 10
alors
  ajouter 10 à i
sinon
  si i > 5
  alors
    ajouter 5 à i
  sinon
    si i > 1
    alors
      ajouter 1 à i
    sinon
      oter 1 à i
    fin_si
  fin_si
fin_si

»

Organigramme



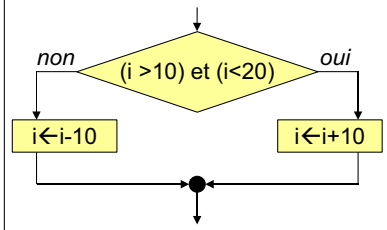
Python

```
if (i>10):
   i = i+10
elif (i>5):
   i = i+5
elif (i>1):
   i = i+1
else:
   i = i-1
```

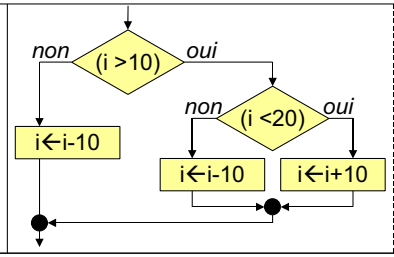
III.B.5. Les conditions composées

- Conditions construites avec
  - Plusieurs conditions plus simples
  - Reliées par des opérateurs logiques
- Opérateurs logiques :
  - La négation, notée « not(...) »
  - Le "et", noté « … and ... »
  - Le "ou inclusif", noté « … or ... »

III.B.6.a. Test avec condition composée "et"

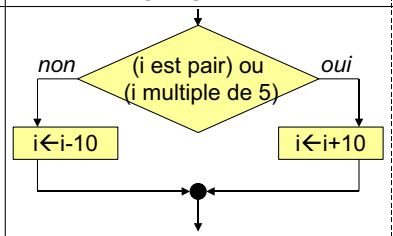
	Organigramme	Python
si i est supérieur à 10 *et* i est inférieur à 20 **alors** ajouter 10 à i **sinon** oter 10 à i **fin_si**		**if ((i>10) and (i<20)):** **i = i+10** **else:** **i = i-10**

➔ IMPORTANT : avec « and » on teste les deux conditions (i>10) et (i<20).

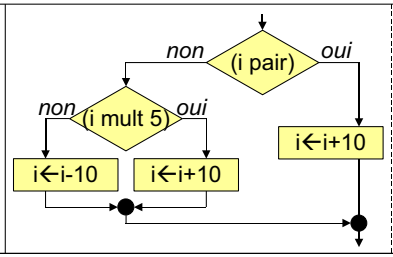
si i est supérieur à 10 *et si en plus* i est inférieur à 20 **alors** ajouter 10 à i **sinon** oter 10 à i **fin_si**		**if (i>10):** **if(i<20):** **i = i+10** **else:** **i=i-10** **else:** **i = i-10**

➔ IMPORTANT : pas de court-circuit, il faut faire 2 tests

III.B.6.b. Test avec condition composée "ou"

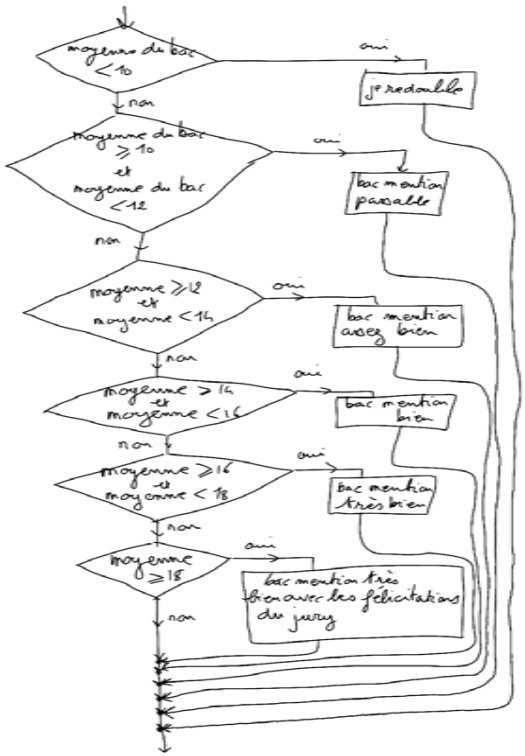
	Organigramme	Python
si i est pair *ou* i est multiple de 5 **alors** ajouter 10 à i **sinon** oter 10 à i **fin_si**		**if ((i pair) or (i mult de 5)):** **i = i+10** **else:** **i = i-10**

➔ IMPORTANT : avec « or » on teste les deux conditions i pair et i mult de 5

si i est pair *ou si par ailleurs* i est multiple de 5 **alors** ajouter 10 à i **sinon** oter 10 à i **fin_si**		**if (i pair)** **i = i+10** **else** **if(i mult de 5):** **i=i+1** **else** **i = i-10**

➔ IMPORTANT : pas de court-circuit, il faut faire 2 tests

III.B.6. Autre exemple

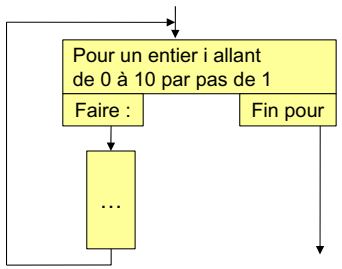


III.C. Les boucles

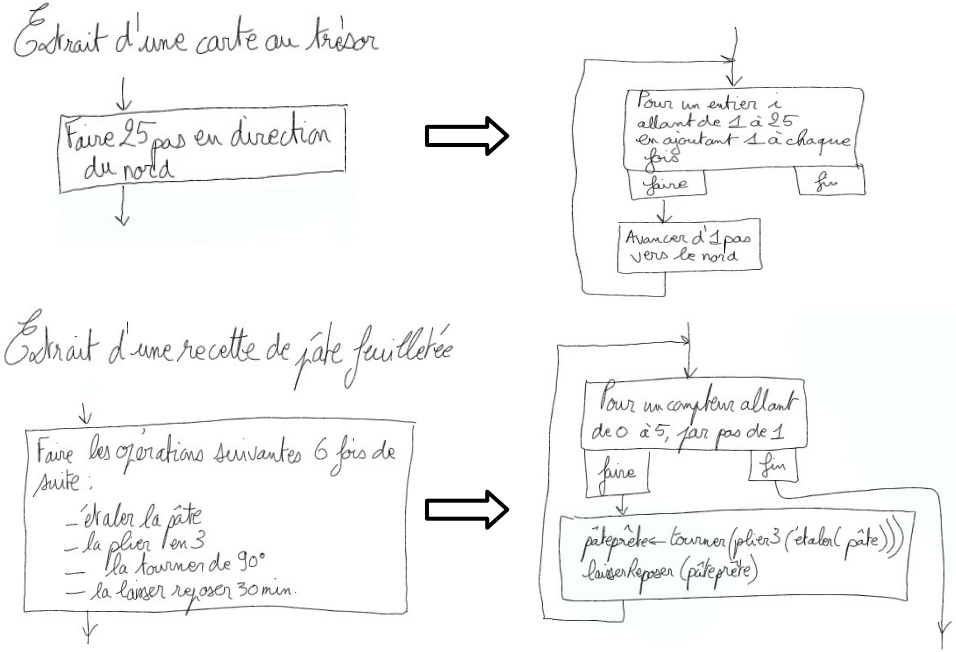
III.C.1. Les 2 types de boucles

- Pour ... Faire ...
- Tant que ... Faire ...
- Vocabulaire :
  - Une itération : un passage dans la boucle
  - Le test (la condition) d'arrêt : détermine la sortie de la boucle

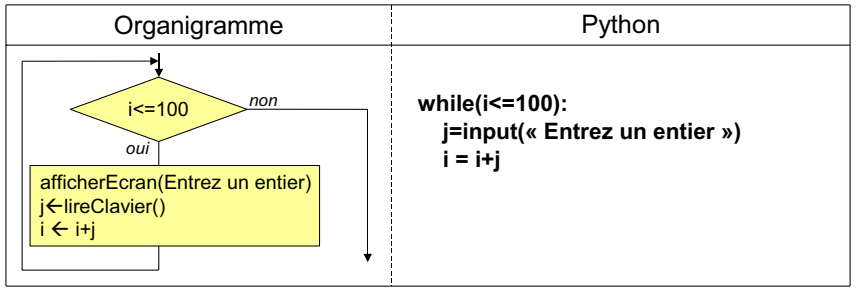
III.C.1.a. Boucle « pour … faire … »

Organigramme	Python
	**for i in range(0,10,1):** **…**

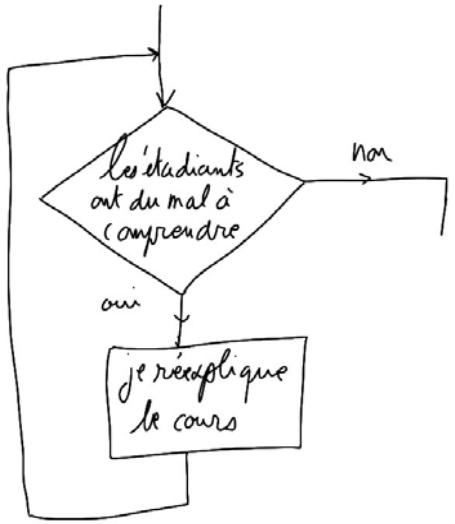
III.C.1.a. Boucle « pour … faire … »



III.C.1.b. Boucle « tant que … faire … »



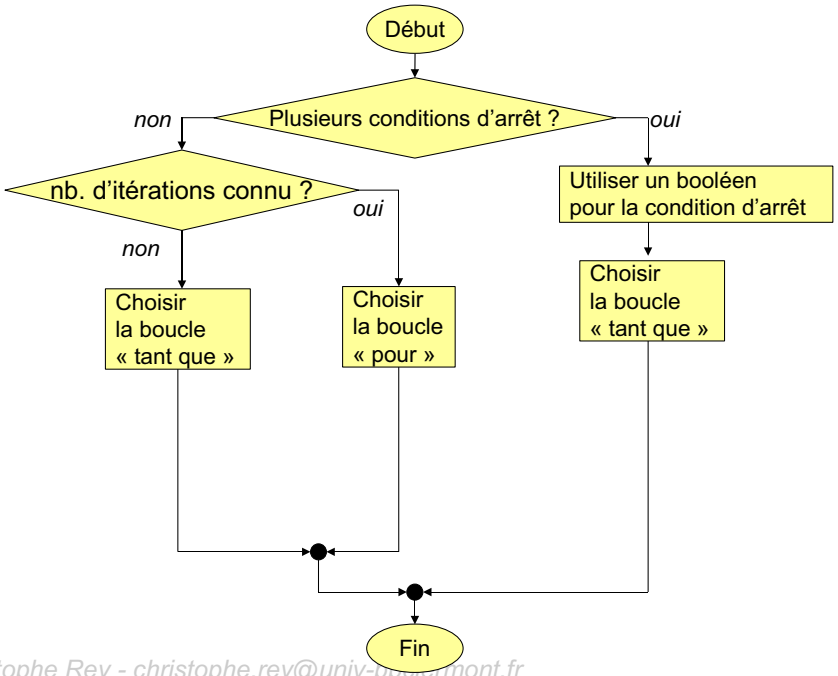
III.C.1.b. Autres exemples



« tant que les étudiants ont du mal à comprendre, je réexplique le cours »

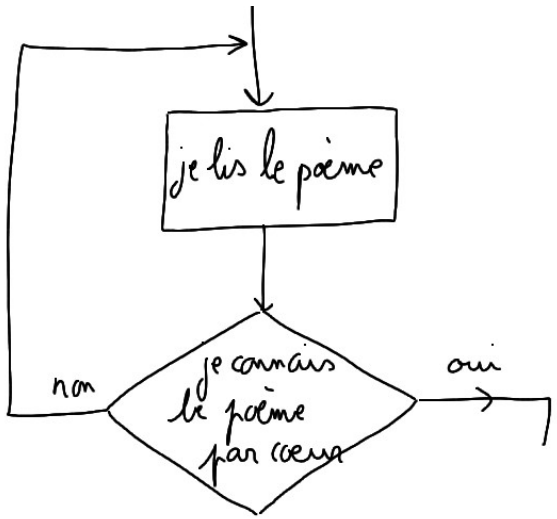
« tant qu'Ulysse n'est pas rentré sur son île d'Ithaque, Pénelope défait son ouvrage la nuit et le refait le jour »

III.C.1.d. Choix d'un type de boucle

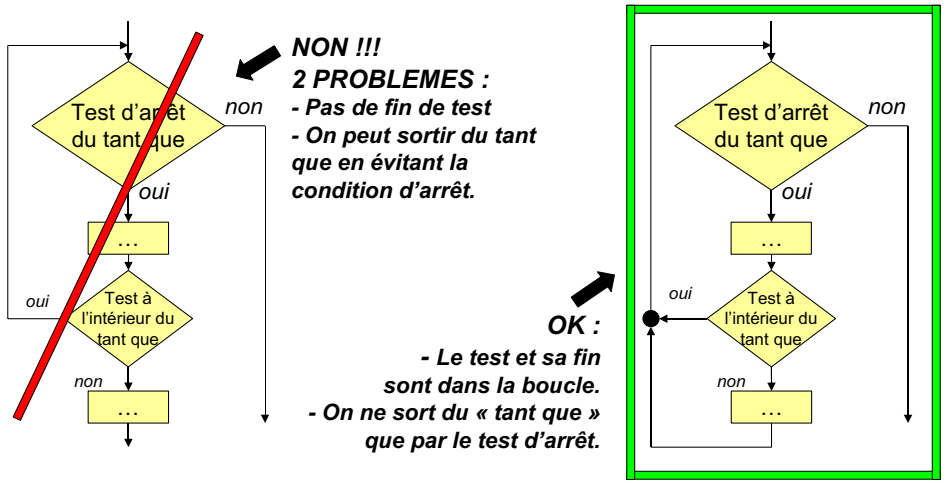


III.C.1.d. Exemple

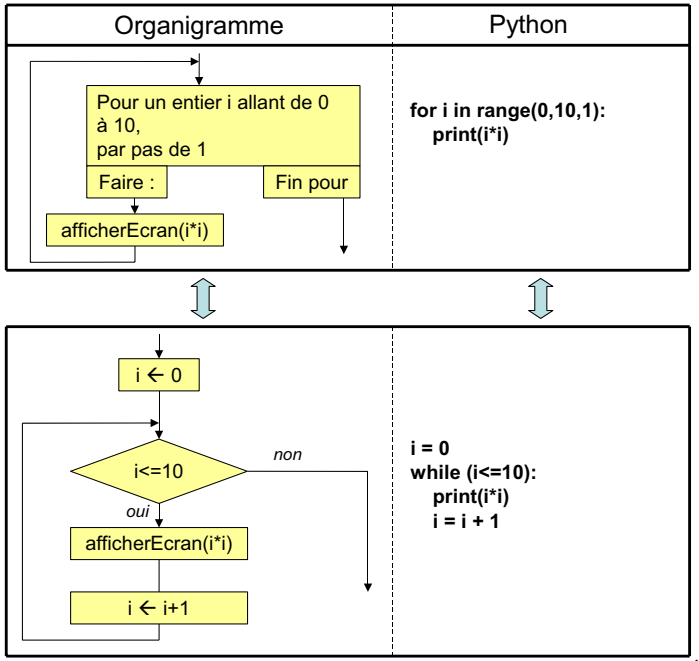
« tant que je ne connais pas le poème par coeur, je le relis »



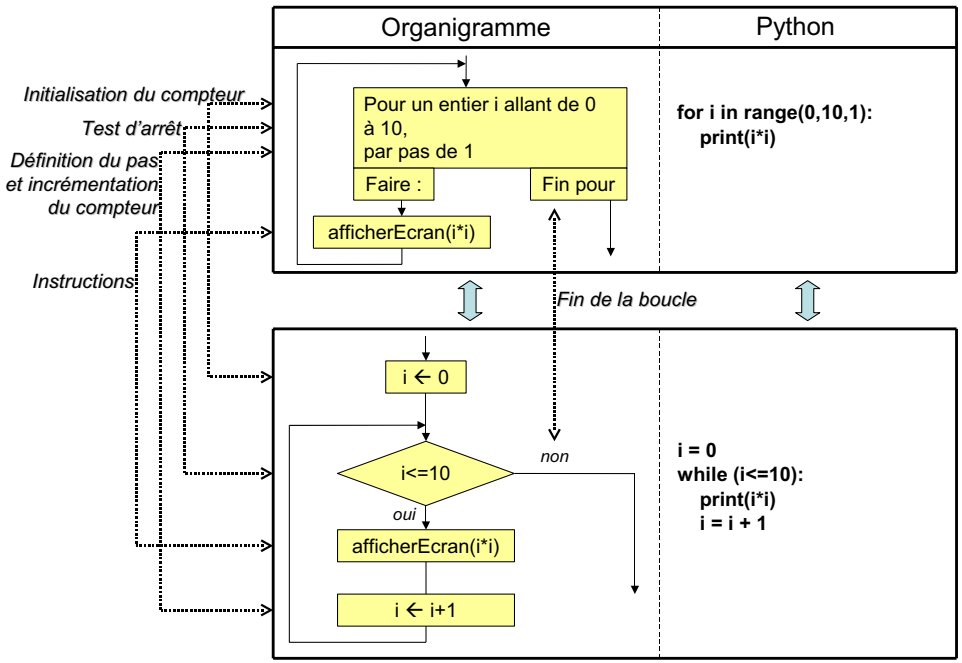
III.C.2. La sortie des boucles



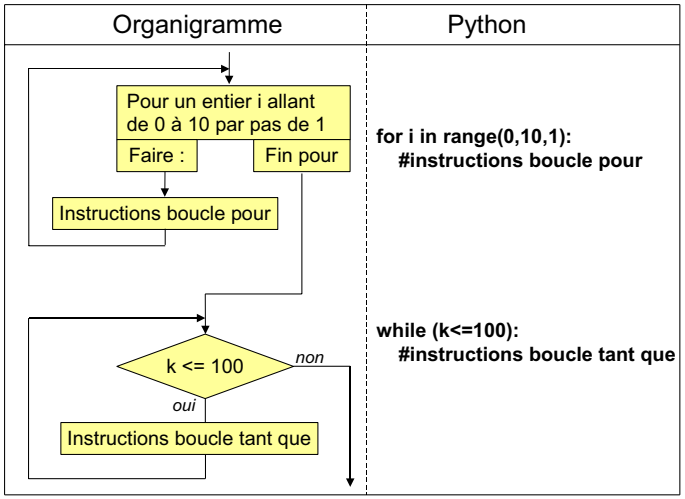
III.C.3.a. « pour » exprimé avec « tant que »



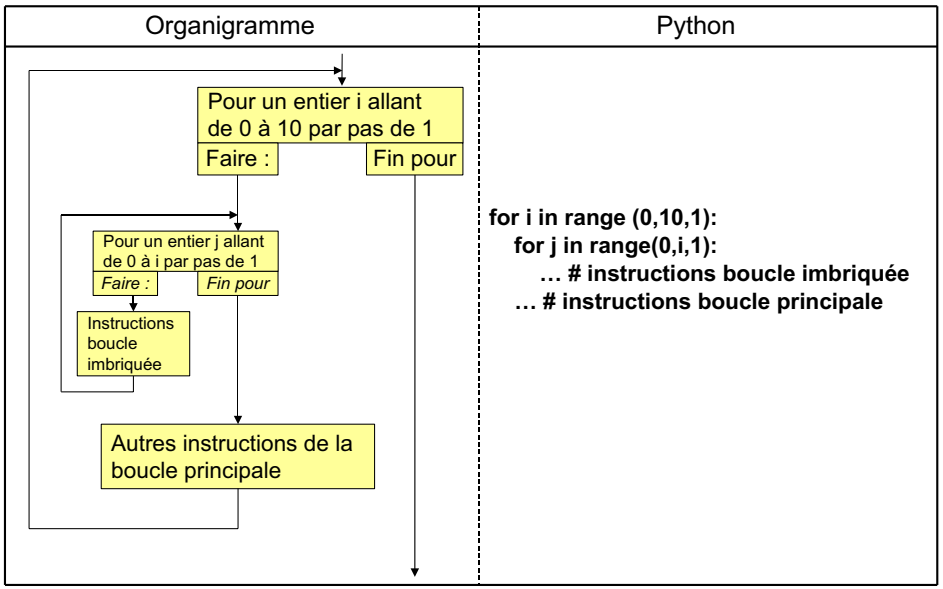
III.C.3.a. « pour » exprimé avec « tant que »



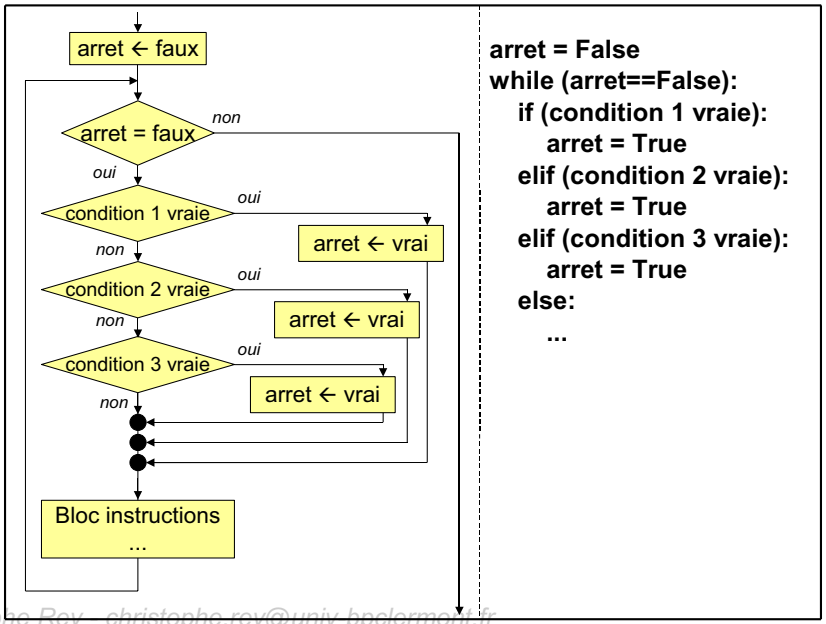
III.C.4.a. Boucles successives



III.C.4.b. Boucles imbriquées « pour »

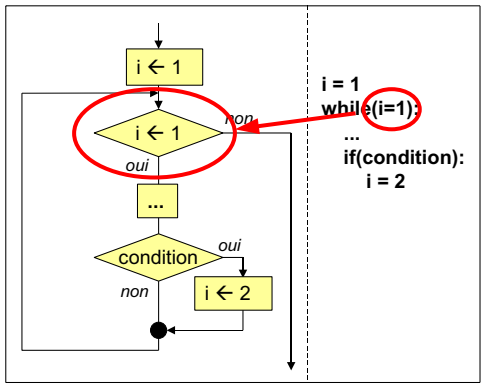


III.C.5. "tant que" avec 3 conditions d'arrêt

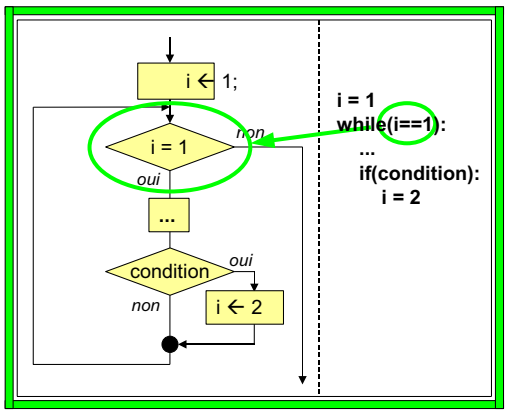


III.C.6.a. Erreurs à ne pas faire

Boucle infinie : confusion = et ==



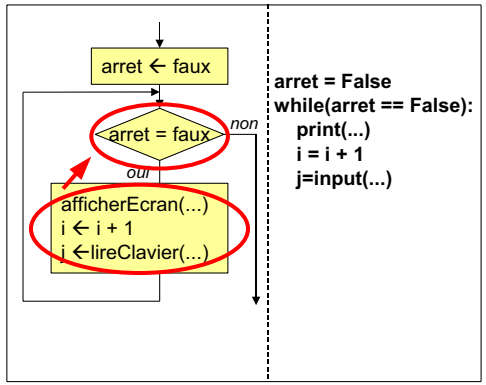
ERREUR : L'affectation est confondue avec la comparaison d'égalité. Le test serait toujours vrai et la boucle ne s'arrêterait jamais.
Python détecte cette erreur.



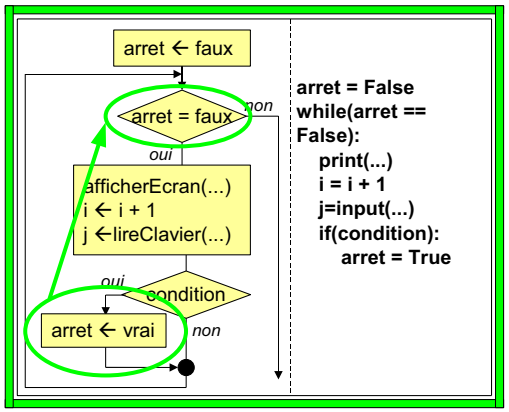
OK : l'erreur a été rectifiée.

III.C.6.a. Erreurs à ne pas faire

Boucle infinie : condition d'arrêt non modifiée



ERREUR : Aucune instruction dans le bloc de la boucle "tant que" ne modifie la variable arret qui contrôle l'arrêt de la boucle.
Python ne détecte pas cette erreur.



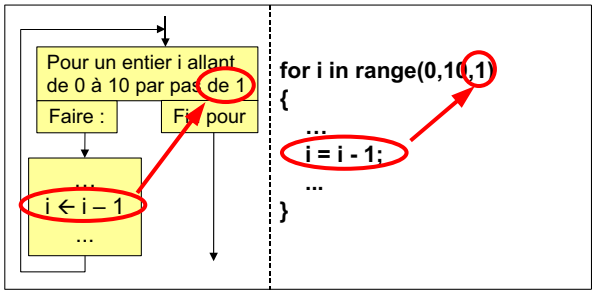
OK : l'erreur a été rectifiée.

III.C.6.b. Erreurs à ne pas faire

Modification dangereuse du compteur d'une boucle "pour"

ERREUR : Le compteur i de la boucle "pour" est modifié intempestivement dans le bloc d'instructions de la boucle.

Ici, comme on enlève 1 (dans le bloc) et on ajoute 1 (dans l'entête de la boucle "pour"), globalement, i ne varie pas et on est dans un cas de boucle infinie.

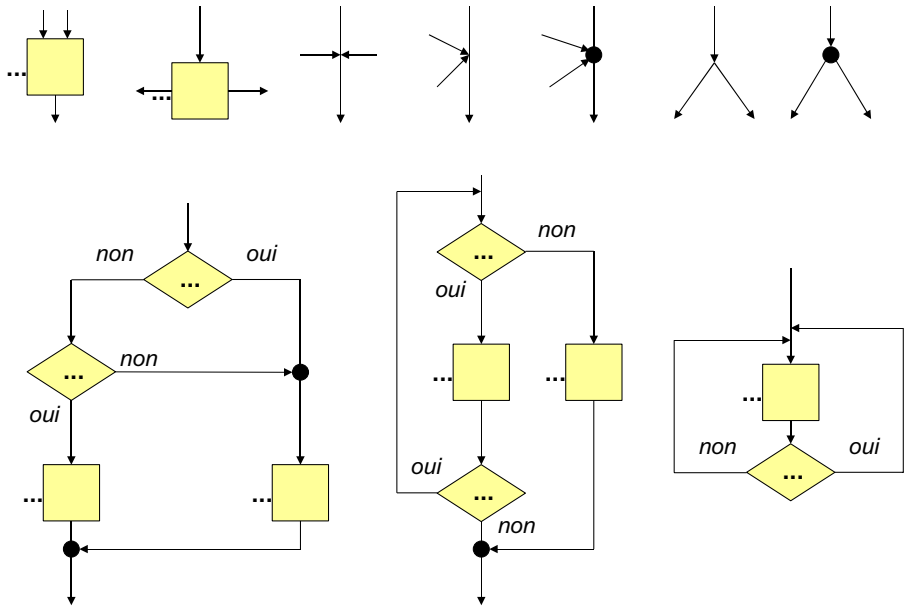


cas de boucle infinie

III.D. JAMAIS de branchements

Jamais de GOTO

III.E. Autres erreurs à ne pas faire



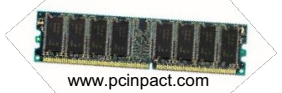
IV. Introduction à la programmation structurée avec le langage Python

La programmation structurée

- Décomposer une tâche en sous-tâches (réaliser une fonction par sous-tâche)
- Décrire chaque fonction (en Python) en suivant les règles de l'algorithmique
- Gérer la mémoire (variables et types nécessaires)

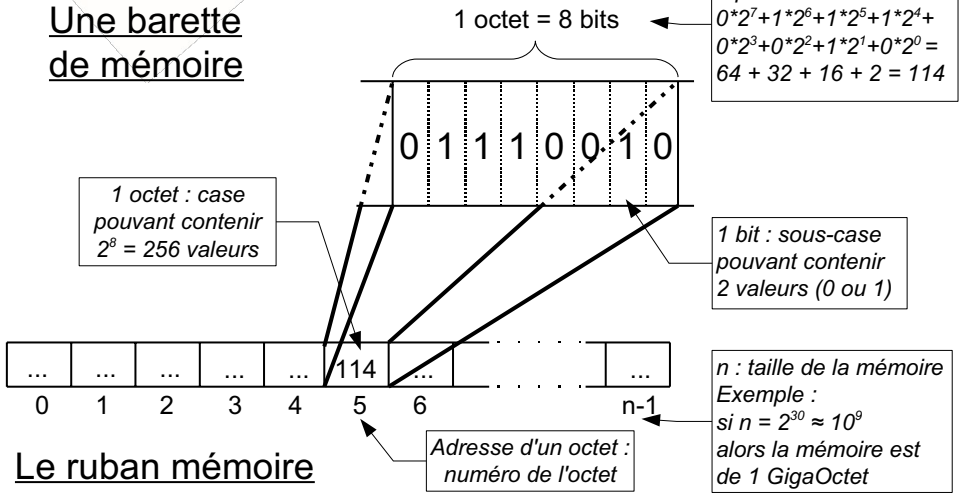
IV.A. Gestion de la mémoire

IV.A.1. Ruban mémoire, bit et octet



www.pcinpact.com

Une barette de mémoire

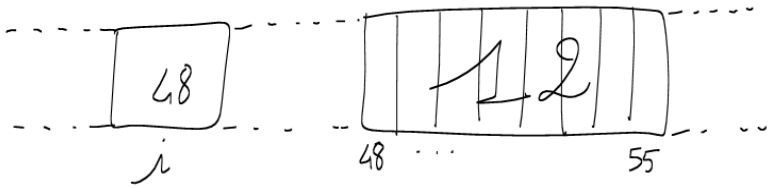


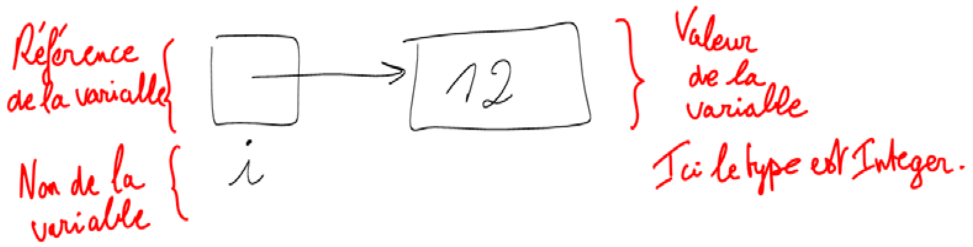
Le ruban mémoire

IV.A.2. Une variable, c'est :

- Un espace mémoire réservé (on dit aussi alloué) de 1 ou plusieurs octets contigüs
- Un type :
  - Le nombre d'octets
  - La nature des valeurs stockées (entier, réel, caractère, adresse)
- Une référence (ou adresse) : adresse du premier octet
- Un nom : pour manipuler la variable
- Un contenu : la valeur stockée

IV.A.2. Représenter une variable





IV.A.3. Notion de type

- Ambiguité: variable = contenant ou contenu
- Le type est ce qui caractérise le contenant.
- Exemple culinaire :
  - on veut faire cuire un poulet
  - on ne va pas le mettre dans un coquetier (!!!)
    - trop petit
    - pas adapté à la cuisson
- Exemple de programmation :
  - on veut mettre un entier dans une variable
  - il faut que le type de la variable soit le bon
    - assez grand
    - configuré pour stocker des valeurs entières (et pas à virgule)

IV.A.3. Les types en Python

Catégorie de types	Type	Exemple
Types standard	boolean	`f=True` `condition=False`
	integer	int: 10, 100, -786, 080, -0490, -0x260, 0x69 long: 51924361L, -0x19323L, 0122L, 0xDEFABCECBDAECBFBAEl, 535633629843L, -052318172735L, -4721885298529L float: 0.0, 15.20, -21.9, 32.3+e18, -90., -0.0, 70.2-E12 complex: 3.14j, 45.j, 9.322e-36j, .876j, -.6545+0J, 3e+26J, 4.53e-7j
	long	
	float	
	complex	
	string	`s="Bonjour tout le monde !"`

	list	`list1 = ['physics', 'chemistry', 1997, 2000];`
		`list2 = [1, 2, 3, 4, 5 ];`
		`list3 = ["a", "b", "c", "d"];`

	tuple	`tup1 = ('physics', 'chemistry', 1997, 2000)`
		`tup2 = (1, 2, 3, 4, 5 )`
		`tup3 = "a", "b", "c", "d"`

	dictionary	`dict1 = { 'abc': 456 };` `dict2 = { 'abc': 123, 98.6: 37 };`
Types personnalisés	object	Cf après

==> Types atomiques VS types containers

IV.A.4. Notion de type container

- Pouvoir stocker dans une variable plusieurs valeurs
- Exemples culinaires
  - un plateau repas :
    - un plat
    - une entrée
    - un dessert
    - un morceau de pain
  - une plaque de cuisson
    - 50 biscuits

En programmation
une classe :
un entier (int)
un booléen (boolean)
un réel (double)
un caractère (char)
un tableau d'entiers
50 entiers (long)

IV.A.4. Notion de type container

- Pouvoir stocker dans une variable plusieurs valeurs
- Exemples culinaires
  - un plateau repas :
    - un plat
    - une entrée
    - un dessert
    - un morceau de pain
  - une plaque de cuisson
    - 50 biscuits

En programmation

une classe :
- un entier (int)
- un booléen (boolean)
- un réel (double)
- un caractère (char)

un tableau d'entiers
- 50 entiers (long)

des valeurs de types différents

IV.A.4. Notion de type container

- Pouvoir stocker dans une variable plusieurs valeurs
- Exemples culinaires
  - un plateau repas :
    - un plat
    - une entrée
    - un dessert
    - un morceau de pain
  - une plaque de cuisson
    - 50 biscuits

En programmation

une classe :
- un entier (int)
- un booléen (boolean)
- un réel (double)
- un caractère (char)

un tableau d'entiers
- 50 entiers (long)

des valeurs du même type

IV.A.4. Notion de type container

- Pourquoi des types container ?
  - Éviter de manipuler des dizaines de variables
  - Faire des traitements répétitifs
  - Regrouper des données qui sont liées
  - Possibilité pour les fonctions de renvoyer plusieurs résultats !
- Containers de données de même type : listes, tuples
- Containers de données de types différents : classes, listes

IV.A.4. Les classes

But Définir des types personnalisés regroupant dans une même variable plusieurs autres sous-variables de types différents.

Syntaxe

```
class MaClassePersonnalise:
    "Classe qui contient ..."
```

Les variables qui seront de ce type, on dit « les objets de cette classe », seront composées de sous-variables dits « attributs ».

Exemples

```
class Date:
    "…"

d=Date()
d.jour=12
d.mois=2
d.annee=2011
```

```
class Heure:
    "…"

h=Heure()
h.heures=23
h.minutes=34
```

```
class Horaire:
    "..."
ho=Horaire()
ho.date=Date()
ho.heure=Heure()
ho.date.jour=23
ho.date.mois=4
ho.date.mois=2011
ho.heure.heures=15
ho.heure.minutes=3
```

IV.A.4. Terminologie

- Les variables de type personnalisé sont appelées

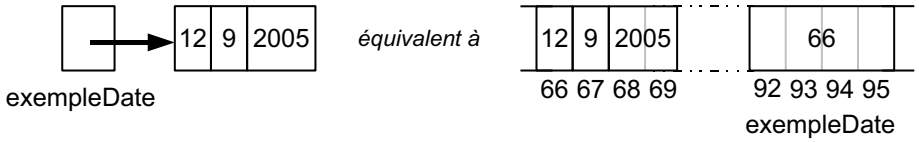
« objets de classe »

- Les sous-variables d'un objet sont appelés

« attributs de l'objet »

IV.A.4. Notion de référence

- Rappel : le nom d'un objet identifie l'espace mémoire qui contient la référence à l'espace mémoire qui contient la valeur



Représentation usuelle *Représentation détaillée*

IV.A.5. Mutable/Immutable

- La valeur des variables des types immutables ne peut être modifiée.
  ==> par contre on peut créer une nouvelle valeur et la lier à un nom de variable déjà existant.
  Types concernés
  - String
  - Tuples
  - Numbers
- Les sous-variables de certains types container peuvent être modifiées :
  - Listes
  - Objects
  - Dictionaries

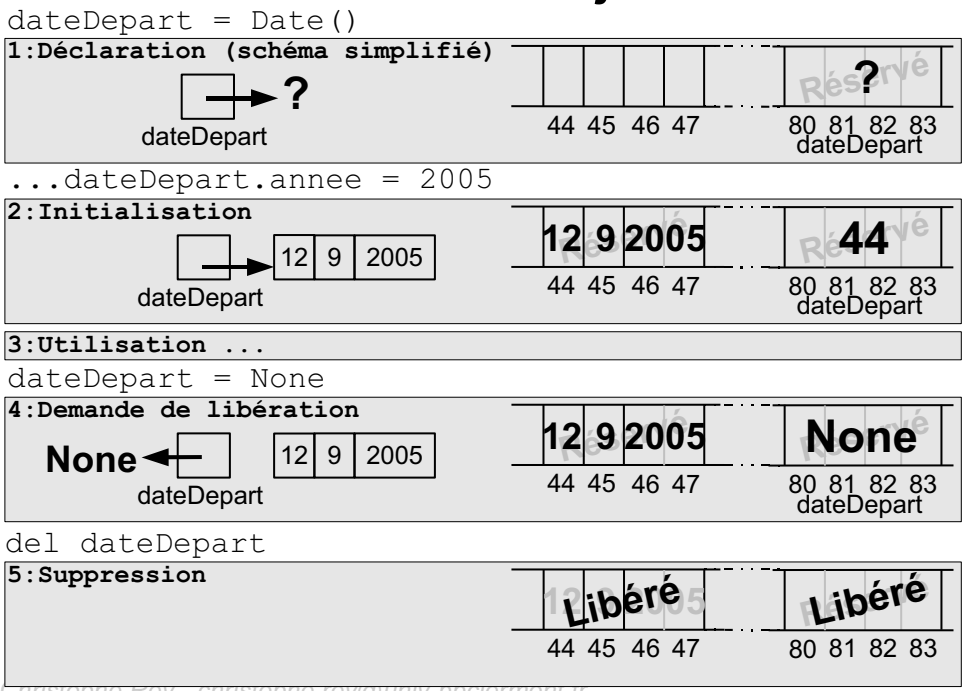
IV.A.5.a. Variables de type atomique

- Définition (création) :
  - Déclaration :
    quel type, quel nom de variable
  - Allocation : réservation mémoire
  - Initialisation : première valeur
    ==> en une seule étape
    ==> le programmeur ne spécifie pas explicitement le type
- Exemples :
  i=23
  s= "Bonjour !"

IV.A.5.b. Variables objets de classes

```
# 1: déclaration, allocation et initialisation
# de la référence de l'objet
dateDepart = Date()

# 2: initialisation des attributs de l'objet
dateDepart.jour = 12
dateDepart.mois = 9
dateDepart.annee = 2005

# 3: utilisation...

# 4: demande de libération
dateDepart = None

# 5: suppression de la variable
del dateDepart
```


IV.A.5.b. Variables objets de classes

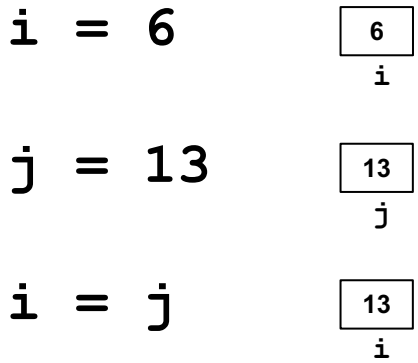


IV.A.6. Manipulation des variables

- Principe :
  - seules les adresses peuvent être modifiées
  - manipulation par référence :
    les adresses sont copiées et on est censé travailler sur les valeurs originales (on ne copie jamais les valeurs)
- Résultat
  - Impression de manipulation par valeur pour les types immutables (numbers, strings, tuples)
    comme si on travaillait sur des copies des valeurs originales
  - Impression de manipulation par référence pour les types « mutables » (Listes, Objets, Dictionnaires) ceux dont les sous-valeurs) sont liées par des adresses.

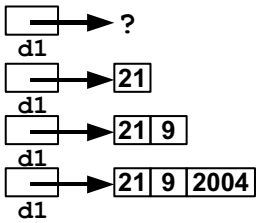
IV.A.6.a. Affectation

Cas des variables de type immutable : schéma simplifié

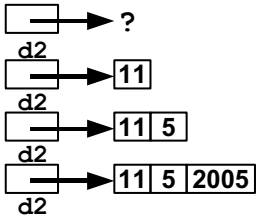


IV.A.6.a. Affectation, type mutables

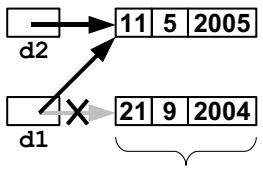
```
d1 = Date()
d1.jour = 21
d1.mois = 9
d1.annee = 2004
```




```
d2 = Date()
d2.jour = 11
d2.mois = 5
d2.annee = 2005
```



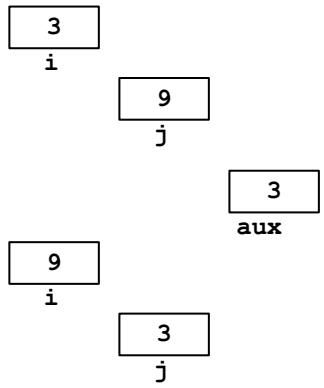

```
d1 = d2
```



La référence de cet objet n'est stockée nulle part : l'espace occupé va être libéré par un mécanisme du type ramasse-miettes.

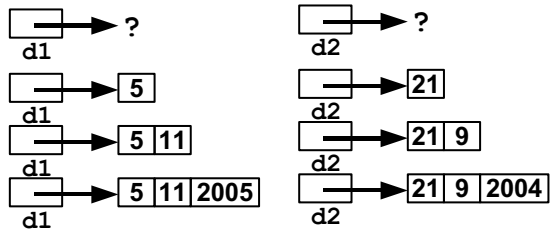
IV.A.6.b. Echange, types immutables

```
i=3
j=9
aux = i
i = j
j = aux
```

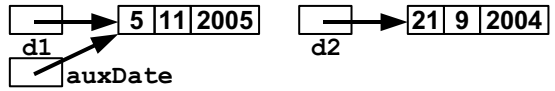


IV.A.6.b. Echange, types mutables

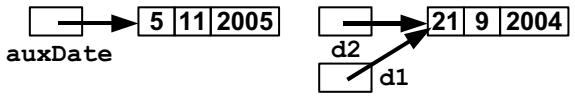
```
d1 = Date()
d1.jour = 5
d1.mois = 11
d1.annee = 2005
d2 = Date()
d2.jour = 21
d2.mois = 9
d2.annee = 2004
```



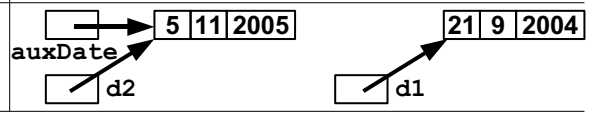

```
auxDate = d1
```




```
d1 = d2
```




```
d2 = auxDate
```



IV.A.7. Portée des variables

- Durée de vie = la fonction dans laquelle elles ont été définies
- Les fonctions peuvent utiliser les variables des fonctions dans lesquelles elles sont définies, en lecture seulement.
  ==> à EVITER
- Les variables globales
  ==> possible, mais à EVITER

IV.B. Fonctions

IV.B.1. Fonctions - principe

- Regrouper des instructions en un tout cohérent
- Représenter les grandes étapes, ou les tâches les plus importantes d'un programme principal
- But :
  - « diviser » (les difficultés)
    « pour régner » (résoudre les problèmes plus facilement)
  - Faciliter la compréhension
  - Faciliter la réutilisation

IV.B.1. Exemple

Résumé : Structure d'un programme Python type

```
####################################
# Programme Python type            #
# auteur : G.Swinnen, Liège, 2003  #
# licence : GPL                    #
####################################


####################################
# Importation de fonctions externes :

from math import sqrt


####################################
# Définition locale de fonctions :


def occurrences(car, ch):
    "nombre de caractères <car> \
     dans la chaîne <ch>"

    nc = 0

    i = 0

    while i < len(ch):

        if ch[i] == car:
            nc = nc + 1

        i = i + 1

    return nc


##############################
# Corps principal du programme :

print "Veuillez entrer un nombre : "
nbr = input()

print "Veuillez entrer une phrase : ",
phr = raw_input()
print "Entrez le caractère à compter : ",
cch = raw_input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print "La racine carrée du cube",
print "du nombre fourni vaut",
print rc
print "La phrase contient",
print no, "caractères", cch
```

Un programme Python contient en général les blocs suivants, dans l'ordre :
- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales)
- Les définitions locales de fonctions et/ou de classes
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes. (Vous pouvez vous-même définir de tels modules).

La définition d'une fonction comporte souvent une liste de paramètres : ce sont toujours des variables, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit en principe inclure les 4 éléments suivants :
- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incrémentation du compteur.

La fonction 'renvoie' toujours une valeur bien déterminée au programme appelant. (Si l'instruction return n'est pas utilisée, ou si elle est utilisée sans argument,la fonction renvoie un objet vide : <None>)

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'arguments, lesquels peuvent être des valeurs, des variables, ou même des expressions.

IV.B.1. Exemple

```
def occurrences(car, ch):
    "nombre de caractères <car> \
     dans la chaîne <ch>"

    nc = 0

    i = 0

    while i < len(ch):

        if ch[i] == car:
            nc = nc + 1

        i = i + 1

    return nc


##############################
# Corps principal du programme :

print "Veuillez entrer un nombre : "
nbr = input()

print "Veuillez entrer une phrase : ",
phr = raw_input()
print "Entrez le caractère à compter : ",
cch = raw_input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print "La racine carrée du cube",
```

La définition d'une fonction comporte souvent une liste de paramètres : ce sont toujours des variables, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit en principe inclure les 4 éléments suivants :
- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incrémentation du compteur.

La fonction 'renvoie' toujours une valeur bien déterminée au programme appelant. (Si l'instruction return n'est pas utilisée, ou si elle est utilisée sans argument,la fonction renvoie un objet vide : <None>)

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'arguments, lesquels peuvent être des valeurs, des variables, ou même des expressions.

IV.B.3. Appel de fonction

- Principe :
  - Utiliser une fonction dans une autre fonction
  - Sans recopier le code de la fonction utilisée
- Fonction
  - appelante : celle qui appelle l'autre
  - appelée : celle qui est appelée
- Appel
  - Appel de fonction : avec affectation
  - Appel de procédure : sans affectation

IV.B.3.a. Paramètres formels/d'appel

- « paramètre » = « argument »
- Paramètre formel :
  - Dans l'entête de la fonction
  - Le nom du paramètre qui sera utilisé dans la définition de la fonction
  - Le type du paramètre
- Paramètre d'appel :
  - Dans l'un appel de la fonction par une autre fonction
  - Soit une valeur (par exemple : 23, 'a', ...)
  - Soit le nom d'une variable de la fonction appelante. (en général différent du nom du paramètre formel.)
  - On ne précise pas le type.

IV.B.3.b. Passage de paramètres

IMPORTANT !!!

- Principe :
  quand une fonction est appelée, tous les paramètres d'appel sont copiés et les instructions de la fonction appelée s'effectuent sur **les copies des paramètres d'appel**
- 2 types de passage :
  - Passage par valeur (abus de langage)
    - Paramètres de **type immutable**
    - Copie de la valeur du paramètre d'appel
  - Passage par référence (ou adresse)
    - Paramètres **de type mutable**
    - Copie de la référence du paramètre d'appel

IV.B.3.b. Conséquences

IMPORTANT !!!

- Passage par valeur
  Copie de la valeur du paramètre d'appel
  => quelle que soient les modifications apportées à la valeur du paramètre d'appel, après la fin de l'exécution de la fonction appelée le paramètre d'appel reprend sa valeur d'avant l'appel de la fonction
- Passage par référence (ou adresse)
  Copie de la référence du paramètre d'appel
  => toutes les modifications apportées à l'objet subsistent après la fin de l'exécution de la fonction appelée
  => par contre, à la fin de la fonction, la référence à l'objet reprend sa valeur d'avant l'appel de la fonction

IV.B.3.b. Passage par valeur Exemple

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```


IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

Début du programme : la fonction __main__ est exécutée.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35

k

Définition de la variable k (déclaration et allocation mémoire), suivie de son initialisation à la valeur 35.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35
k

Appel de la fonction ajoute10 par la fonction main.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35
k

k est le paramètre d'appel.
C'est le paramètre situé dans l'appel de la fonction.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35
k

i est le paramètre formel correspondant à k.
C'est le paramètre situé dans la définition de la fonction.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Est copiée dans

Variables de la fonction ajoute10

35
i = copie de k

Variables de la fonction main

35
k

Le type de k est immutable : le passage est par valeur.
i va donc contenir une copie de la valeur de k.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

45
i

Variables de la fonction main

35
k

La valeur stockée dans la copie de k (nommée i) passe à 45.
La valeur stockée dans k (l'originale) reste à 35.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

45
i

Variables de la fonction main

35
k

?
j

La valeur renvoyée par la fonction est celle stockée dans i, c'est-à-dire 45.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)45
```

Variables de la fonction ajoute10

45
i

Valeur retournée : 45

Variables de la fonction main

35
k

C'est comme si, dans le code, on remplaçait l'appel de la fonction par la valeur résultat.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

45
i

Variables de la fonction main

35
k

45
j

La valeur résultat (45) est ensuite affectée à j.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35
k

45
j

A la fin de l'exécution de la fonction ajoute10, les copies des paramètres et les variables locales sont effacées (libérées).

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

35
k

45
j

On remarque que la valeur de k n'a pas changé : elle vaut toujours 35.

IV.B.3.b. Passage par valeur - Ex.

```
def ajoute10(i):
    i = i + 10
    return i

k = 35
j = ajoute10(k)
```

Variables de la fonction ajoute10

Variables de la fonction main

A la fin de la fonction main (la fin du programme principal), les variables locales k et j sont à leur tour libérées.

IV.B.3.b. Passage par référence Exemple

IV.B.3.b. Passage par référ. - Ex.

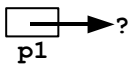
```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```


IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



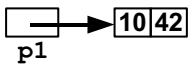
Déclaration et allocation d'un objet p1 de classe Point.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



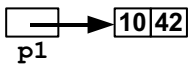
Initialisation des attributs de p1.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



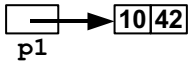
Appel de la fonction ajoute10.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10( p ):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10( p1 )
```

Variables de la fonction ajoute10

Variables de la fonction main



Le paramètre d'appel est p1. Le paramètre formel est p.

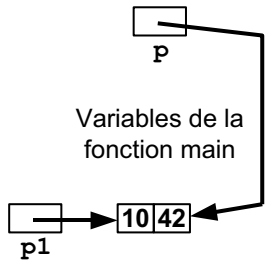
IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10( p ):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10( p1 )
```

La référence p1 est copiée dans

Variables de la fonction ajoute10

Variables de la fonction main



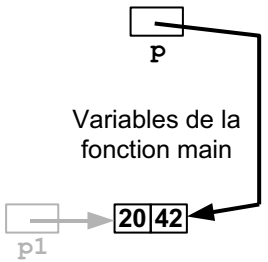
p1 est un objet : le passage est par référence. p va donc contenir une copie de p1 qui est la référence à l'objet.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



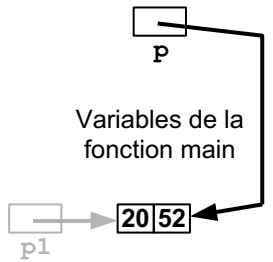
10 est ajouté à p.x c'est-à-dire aussi à p1.x puisque l'objet est le même.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



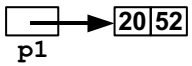
10 est ajouté à p.y c'est-à-dire aussi à p1.y puisque l'objet est le même.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main



A la fin de l'appel à ajoute10, p est libéré, mais l'objet de référence p1 conserve les modifications.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main

A la fin de la fonction main, p1 est libérée car elle avait été déclarée dans la fonction main.

IV.B.3.b. Passage par référ. - Ex.

```
class Point:
      "contenant 2 attributs x et y"
def ajoute10(p):
    p.x = p.x + 10
    p.y = p.y + 10

p1 = Point()
p1.x = 10
p1.y = 42
ajoute10(p1)
```

Variables de la fonction ajoute10

Variables de la fonction main

Avant que le programme ne soit totalement fini, le système libère tous les objets restants.

IV.B.3.c. Valeur de retour

- Principe :
  - La valeur de retour est l'adresse de la variable qui suit l'instruction return.
- Pas de différence entre types mutables et immutables
- Unicité de la valeur de retour
  mais possibilité de retourner plusieurs valeurs en retournant un (seul) objet

IV.B.3.c. Retour d'un objet
Exemple

IV.B.3.c. Retour d'un objet

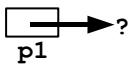
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```


IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10

Variables de la fonction main



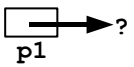
Déclaration et allocation d'un objet p1 de classe Point dans la fonction main.

IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10

Variables de la fonction main

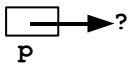


Appel à la fonction créerPoint (pas de paramètre).

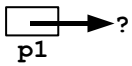
IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10



Variables de la fonction main

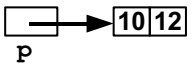


Déclaration et allocation d'un objet p de classe Point dans la fonction créerPoint.

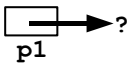
IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10



Variables de la fonction main

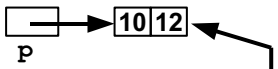


Initialisation des attributs de p.

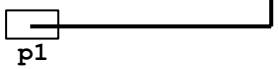
IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10



Variables de la fonction main

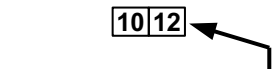


Copie de p dans la variable de la fonction __main__.

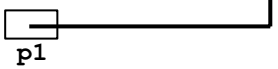
IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10



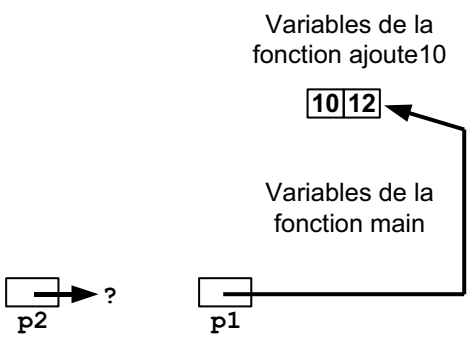
Variables de la fonction main



Libération de p. Les objets référencés restent alloués.

IV.B.3.c. Retour d'un objet

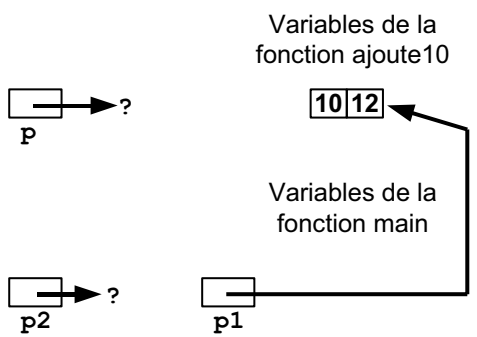
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Déclaration d'un objet p2 de classe Point.

IV.B.3.c. Retour d'un objet

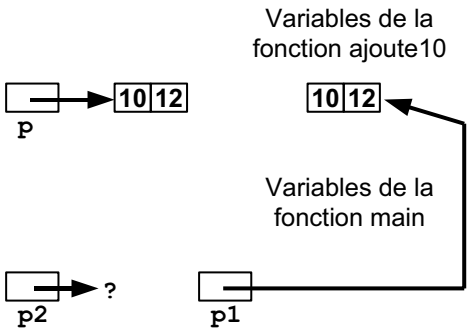
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Déclaration et allocation d'un objet p de classe Point dans la fonction créerPoint.

IV.B.3.c. Retour d'un objet

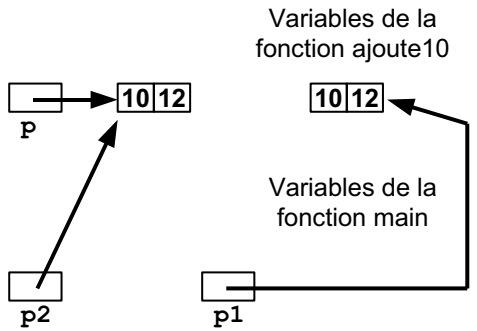
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Initialisation des attributs de p.

IV.B.3.c. Retour d'un objet

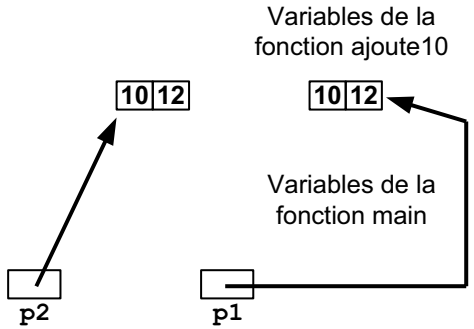
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Copie de p dans la variable de la fonction __main__.

IV.B.3.c. Retour d'un objet

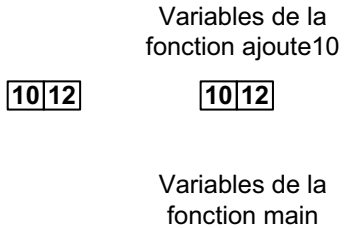
```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Fin de créerPoint : p est libéré mais pas l'objet.

IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```



Fin de __main__ : p1 et p2 sont libérés en premier...

IV.B.3.c. Retour d'un objet

```
class Point:
  "contenant les attributs x et y"
def créerPoint():
    p = Point()
    p.x = 10
    p.y = 12
    return p

p1 = Point()
p1 = créerPoint()
p2 = créerPoint()
```

Variables de la fonction ajoute10

Variables de la fonction main

… puis les objets non référencés sont libérés par le système.

V. Introduction à la programmation orientée objet

V.A. Notions de base

V.A. Une nouvelle organisation

```
def fonction1(n, x):
    …
def fonction2(x):
    …
def fonction3(n):
    …

f = fonction1(i,d)
b1 = fonction2(d)
b2 = fonction3(i)
```

Fichier Exemple.py

Programmation structurée

```
class Outils:
    "…"
    def fonction1(self):
        …
    def fonction2(self):
        …
    def fonction3(self):
        …
```

Fichier Outils.py

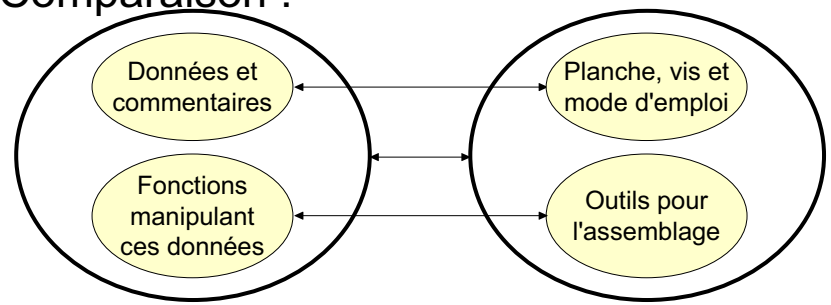
```
o = Outils()

o.n = …
o.d = ...

f = o.fonction1()
b1 = o.fonction2()
b2 = o.fonction3()
```

Fichier Exemple.py

Programmation orientée objet

V.A.1. Principe

- Regrouper
  - les données avec les traitements sur ces données
  - les variables et objets avec les fonctions associées
- Comparaison :



Un objet d'une classe **Un meuble en kit**

V.A.1. Motivations

- Permettre une analyse intuitive des problèmes
  - Modélisation des applications stable dans le temps
  - Construire des applications très complexes
- Obtenir un code plus :
  - Sécurisé
  - Réutilisable
    - Facile à comprendre
    - Rapide à mettre en oeuvre
  - Propre
    - Moins d'erreurs car code plus autonome
    - Possibilités de gérer les erreurs facilement

V.A.1. Définition

- Types personnalisés
  - Permettent de définir des objets avec attributs (i.e. des variables contenant des sous-variables)
- Les classes
  - Sont des types personnalisés avec des méthodes (i.e. Les fonctions permettant de manipuler les attributs.)
  - Permettent
    - L'encapsulation (en théorie, en Python ce n'est pas vraiment possible)
    - Et donc l'abstraction des données
    - L'héritage
    - Le polymorphisme

V.A.1. Exemple

```
##############################################
# Classe exemple Point
class Point:
    """Commentaire obligatoire de description de la classe"""

    def fonction1(self,param1,param2):
    …
    def fonction2(self):
    …
    def affichage(self):
    …
##############################################
# Programme utilisant Point
p=Point()
p.x=3
p.y=4
p.fonction2()
print("Le premier attribut de p est",p.x)
print("p est",p.affichage())
```

Définition des méthodes

Déclaration des attributs

V.A.2. Opérateur point « . »

- Permet l'accès aux attributs et/ou aux méthodes d'un objet
- Exemple
  (avec p un objet de classe Point)
  - « p.x » renvoie la valeur de l'attribut x de l'objet p
  - « p.affichage() » appelle la méthode affichage() pour l'objet p.

V.A.2. Utilisation d'un objet

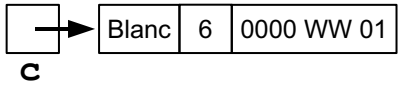
- Un objet doit être utilisé uniquement au travers de ses méthodes.
- Exemple pour l'affichage :
  - Instanciation :

```
c = Clio()
c.couleur = "Blanc"
c.puissance = 6
c.numImm = "0000 WW 01"
```

  - Affichage :

```
print(c)
```

```
class Clio:
    """…commentaires..."""
    def affichage(self):
        …
```



Ne fonctionne pas car c n'est pas une chaine de caractères: affiche l'adresse de c et non les valeurs de ses attributs.

V.A.2. Utilisation d'un objet

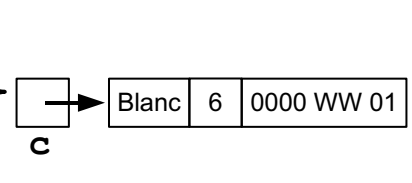
- Un objet doit être utilisé uniquement au travers de ses méthodes.
- Exemple pour l'affichage :
  - Instanciation :

```
c = Clio()
c.couleur = "Blanc"
c.puissance = 6
c.numImm = "0000 WW 01"
```

  - Affichage :

```
print(c)
print(c.couleur, c.puissance, c.numImm)
```

```
class Clio:
    """..."""
    def affichage(self):
        ...
```



Possible, mais peu élégant
car il faut accéder à chaque attribut un par un

V.A.2. Utilisation d'un objet

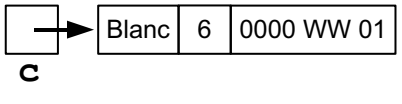
- Un objet doit être utilisé uniquement par ses méthodes.
- Exemple pour l'affichage :
  - Instanciation :

```
c = Clio()
c.couleur = "Blanc"
c.puissance = 6
c.numImm = "0000 WW 01"
```

  - Affichage :

```
print(c)
print(c.couleur, c.puissance, c.numImm)
c.affichage()
```

```
class Clio:
    """..."""
    def affichage(self):
        if(self.puissance!=0):
            print(self.couleur)
            print(self.puissance)
            print(self.numImm)
        else:
            print("données
                  incorrectes")
```



La bonne solution : utiliser la méthode faite pour.

V.A.2. Utilisation d'un objet

- Un objet doit être utilisé **uniquement** au travers de ses méthodes :
  - Éviter les erreurs d'utilisation d'un attributs
  - Être obligé de « bien » utiliser les objets
- => Principe de l'ENCAPSULATION : empêcher l'accès direct aux attributs

V.A.3. Encapsulation en Python

- Impossible à spécifier !!!
- Tous les attributs et méthodes sont accessibles par l'utilisateur d'une classe
  (par ex. dans un programme)
- Encapsulation des données via de bonnes pratiques de programmation

V.A.3. Importance de l'encapsulation

- Encapsulation = interdire l'accès aux attributs en dehors de la classe
- Conséquences :
  - Créer une méthode spécifique pour l'initialisation : le constructeur
  - Créer des accesseurs en lecture et en écriture méthodes setAttribut() et getAttribut()

V.A.3. Exemple

```
class Clio:
    """Ceci est la classe Clio."""

    def affichage(self):
        if(self.puissance!=0):
            print(self.couleur)
            print(self.puissance)
            print(self.numImm)
        else:
            print("""données
                     Incorrectes""")
    def __init__(self,co,pu,nu):
        self.couleur=co
        self.puissance=pu
        self.numImm=nu
```

Constructeur

```
############################
# Programme utilisateur
c=Clio()
c.couleur = "Blanc"
c.puissance = 6
c.numImm = "0000 WW 01"
c.affichage()
```

```
############################
# Programme utilisateur
c=Clio("Blanc",6,
            "0000 WW 01")
c.affichage()
```


V.A.3. Exemple

```
class Clio:
    """Ceci est la classe Clio."""

    def affichage(self):
        if(self.puissance!=0):
            print(self.couleur)
            print(self.puissance)
            print(self.numImm)
        else:
            print("""données
                     Incorrectes""")
    def __init__(self,co,pu,nu):
        self.couleur=co
        self.puissance=pu
        self.numImm=nu
    def getCouleur(self):
        return self.couleur
```

```
############################
# Programme utilisateur
c=Clio("Blanc",6,
            "0000 WW 01")
c.affichage()
print("La couleur est",
          c.getCouleur())
```

Accesseur en lecture pour l'attribut couleur

V.A.3. Exemple

```
class Clio:
    """Ceci est la classe Clio."""

    def affichage(self):
        if(self.puissance!=0):
            print(self.couleur)
            print(self.puissance)
            print(self.numImm)
        else:
            print("""données
                     Incorrectes""")
    def __init__(self,co,pu,nu):
        self.couleur=co
        self.puissance=pu
        self.numImm=nu
    def getCouleur(self):
        return self.couleur
    def setCouleur(self,c):
        self.couleur=c
```

```
############################
# Programme utilisateur
c=Clio("Blanc",6,
            "0000 WW 01")
c.affichage()
print("La couleur est",
          c.getCouleur())
c.setCouleur("Rouge")
```

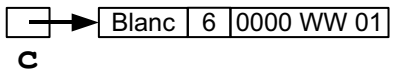
Accesseur en écriture pour l'attribut couleur

V.A.4. Remarques

- On ne peut appeler une méthode que si la référence pointe vers un objet existant.
- None :
  - Signifie l'adresse nulle, égale à 0
  - Peut être affecté à la référence d'un objet pour signifier qu'il n'y a pas d'objet au bout de la référence.
  - Empêche l'utilisation des méthodes, puisqu'il n'y a pas d'objet au bout de la référence.
- Exemple d'utilisation de None :

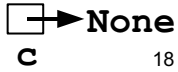
```
c = Clio("Blanc",6, "0000 WW 01")
c.affichage()
```

OK, l'appel à affichage() est possible




```
c = None
c.affichage()
```

NON, l'appel à affichage() n'est pas possible car la référence ne pointe pas un objet existant



V.B. La suite ?

- Concept d'héritage
- Concept de polymorphisme
- La récursivité
- Structures de données
  - Listes
  - Dictionnaires
- Les fichiers, accès aux BD, ..
- ...

Exercice

Equation du 2nd degré

- Ecrire un programme monolithique (sans fonction) qui résout dans R une équation du 2nd degré $ax^2+bx+c=0$
- Ecrire le même programme mais avec 3 fonctions
  - Une qui demande à l'utilisateur d'entrer les coefficients
  - L'autre qui calcule les solutions
  - La troisième qui affiche les solutions
    Vous aurez besoin d'une classe Coef et d'une classe Solution pour ce faire. Pourquoi ?
- Compléter les classes Coef et Solution avec un constructeur et des accesseurs en lecture et écriture
- Répartissez les 3 fonctions précédentes dans les classes Coef et Solution.
- Mettez vos classes Coef et Solution dans des fichiers séparés.