



UNIVERSITÉ  
TOULOUSE III  
PAUL SABATIER



Faculté  
des Sciences  
et d'Ingénierie

# Méthodologie

## Introduction aux Invariants/Variants

Algorithmique

## Problématique

Formaliser la construction d'un algorithme de répétition

## Démarche

Pour un problème dont on nous fournit les exigences, on va répondre aux questions suivantes :

- ▶ Quel est le problème (**analyse**) ?
- ▶ **Comment** construire l'algorithme de la boucle ?
- ▶ Comment **vérifier** que le code développé est correct en répondant aux exigences (invariant et terminaison) ?

## Exemple de référence

- Énoncé et exemples

- Analyse

- Mise au point du code

- Précisions

## Généralisation

- Méthode

- Récapitulatif

## Outil pour la vérification

- L'instruction Assert

- Mise au point du code

- Interprétation des erreurs

- Finalisation

## Exercices

Exemple de référence

### Énoncé

On donne deux entiers naturels  $X$  et  $Y$ . Écrire un algorithme qui calcule le quotient  $q$  (c'est-à-dire  $X//Y$ ) et le reste  $r$  de la division entière (c'est-à-dire  $X\%Y$ ) **en n'utilisant que les opérations  $+$  et  $-$**

### Exemples

$X$	$Y$	$q$	$r$
13	4	3	1
4	17	0	4
0	4	0	0

### Précondition

X et Y entiers,  $X \geq 0$  et  $Y > 0$

### Postcondition (ou Objectif)

q et r sont LES deux entiers tels que  $X = qY + r$  et  $0 \leq r < Y$   
(où X et Y n'ont pas été modifiés)

### Modèle de solution

On s'inspire de l'exemple  $X=13$  et  $Y=4$  :

- ▶ on effectue des soustractions successives de la valeur 4 à 13
- ▶ on obtient la suite de valeurs : 13, 9, 5, 1
- ▶ on s'arrête lorsque la valeur obtenue est inférieure à 4.

Le modèle de solution consiste donc en des **soustractions successives de Y à une copie de X**. L'algorithme s'arrête lorsque l'on ne peut plus effectuer de soustraction. Le nombre de soustractions effectuées est le quotient de la division euclidienne.

### Programmation orientée **Objectif**

On décompose l'**Objectif**  $X == qY + r$  et  $0 \leq r < Y$  en sous-objectifs maîtrisables :

- ▶ **Propriété 1** :  $X == qY + r$  et  $0 \leq r$ .
  - ▶ Cette propriété doit pouvoir être vraie à l'**initialisation** ainsi qu'à chaque **itération**
  - ▶ Cette propriété est nommée **invariant**.
- ▶ **Propriété 2** :  $r < Y$ 
  - ▶ Cette propriété conditionne l'arrêt de la boucle (*On doit sortir de la boucle quand  $r < Y$*  )

Si le programme s'exécute en suivant un chemin qui satisfait la **Propriété 1** et si la boucle termine selon la **Propriété 2** alors l'**Objectif** est atteint.

## Étape 1 : Invariant et Objectif

```
1 # La pré-condition  $X \geq 0$  et  $Y > 0$  doit etre vérifiée
2 # Initialisation
3 q = ?
4 r = ?
5 # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit etre vérifié
6 while ...:
7     # Modification de q et r
8     q = ?
9     r = ?
10    # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit etre vérifié
11 # L'Objectif :  $X == q * Y + r$  et  $0 \leq r < Y$  doit etre atteint
```



## Étape 2 : Objectif et condition de boucle

```
1 # La pré-condition  $X \geq 0$  et  $Y > 0$  doit etre vérifiée
2 # Initialisation
3 q = ?
4 r = ?
5 # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit etre vérifié
6 while r >= Y:
7     # Modification de q et r
8     q = ?
9     r = ?
10    # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit etre vérifié
11 # L'Objectif :  $X == q * Y + r$  et  $0 \leq r < Y$  est atteint
```

## Étape 3 : Initialisation et invariant

```
1 # La pré-condition  $X \geq 0$  et  $Y > 0$  doit être vérifiée
2 # Initialisation
3  $q = 0$ 
4  $r = X$ 
5 # L'Invariant :  $X = q * Y + r$  et  $0 \leq r$  est vérifié
6 while  $r \geq Y$ :
7     # Modification de  $q$  et  $r$ 
8      $q = ?$ 
9      $r = ?$ 
10    # L'Invariant :  $X = q * Y + r$  et  $0 \leq r$  doit être vérifié
11 # L'Objectif :  $X = q * Y + r$  et  $0 \leq r < Y$  est atteint
```

## Étape 4 : Corps de boucle

```
1 # La pré-condition  $X \geq 0$  et  $Y > 0$  doit être vérifiée
2 # Initialisation
3  $q = 0$ 
4  $r = X$ 
5 # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  est vérifié
6 while  $r \geq Y$ :
7     # Modification de  $q$  et  $r$ 
8      $q = q + 1$ 
9      $r = r - Y$ 
10    # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  est vérifié
11 # L'Objectif :  $X == q * Y + r$  et  $0 \leq r < Y$  est atteint
```

### Non-modification des variables initiales

- ▶ Lors de la définition de l'objectif  $X == q * Y + r$  et  $0 \leq r < Y$ , on a précisé que les variables  $X$  et  $Y$  ne sont pas modifiées.
- ▶ Si l'on ne précise pas cette contrainte, le code suivant respecte l'objectif :

```
1 X=eval(input())
2 Y=eval(input())
3 X,q,r = 0,0,0
4 print(q,r)
5 # en effet, on a bien  $X == q * Y + r$  et  $0 \leq r < Y$  !!
```

- ▶ dans l'objectif, on doit avoir les valeurs *initiales* de  $X$  et  $Y$ , d'où la contrainte de ne pas les modifier. On les a donc notées en majuscules.

### Cas d'un objectif non décomposé (1)

```
1 initialiser les variables
2 while not (objectif):
3     modifier les variables
4 # quand on sort de la boucle,
5 # on est sûr que l'objectif est atteint
```

Dans notre exemple :

```
1 q,r = ?,?
2 while not (X==q*Y+r and 0<=r and r<Y):
3     q,r = ?,?
4 # quand on sort de la boucle,
5 # on est sûr que X==q*Y+r and 0<=r and r<Y
```

### Cas d'un objectif non décomposé (2)

- ▶ fonctionne dans certains cas simples, dont la division euclidienne, MAIS...
- ▶ n'est pas efficace : dans notre exemple on re-évalue inutilement  $X == q*Y + r$  à chaque itération.
- ▶ n'est pas généralisable : contre-exemple : minimum  $m$  d'une liste  $L$  de longueur  $l$  :
  - ▶ Objectif :  $m == \min(L)$
  - ▶ Code : `while not(m == min(L))`  
mais pour tester la condition il faut justement calculer le `min` !

## Généralisation

On décompose l'**Objectif** en 2 propriétés

- ▶ L'**Invariant** (Propriété 1), vrai à l'initialisation ainsi qu'à chaque itération.
- ▶ La **Condition d'arrêt** (Propriété 2) de la boucle.

On a donc le modèle d'algorithme suivant :

```
1 # La pré-condition doit etre vérifiée
2 # Initialisation des variables
3 # L'Invariant (Propriété 1) doit etre vérifié
4 while not (Propriété 2):
5     # Modification des variables
6     # L'Invariant (Propriété 1) doit etre vérifié
7 # L'Objectif (Propriétés 1 et 2) doit etre atteint
```



## Méthode basée sur l'invariant

- ▶ Cette méthode a permis de guider l'élaboration de l'algorithme en répondant aux questions suivantes :
  - ▶ quelle condition de boucle utiliser ?
  - ▶ comment initialiser les variables ?
  - ▶ comment modifier les variables dans la boucle ?
- ▶ Il restera de répondre à la question suivante :
  - ▶ comment être sûr que cette boucle n'est pas infinie ?

Outil pour la vérification

## Syntaxe

```
1 assert <expression logique> , " < message d'erreur>"
```

- ▶ a pour effet :
  - si l'expression logique est **fausse**
    - le programme **s'arrête** en affichant le **message d'erreur**
  - sinon** on poursuit l'exécution du programme.

## Avantages & inconvénients

- ▶ Les erreurs sont localisées et cela permet de mettre au point étape par étape, MAIS...
- ▶ Les erreurs dépendent du cas de test
- ▶ L'exécution est ralentie

En utilisant les assert, on décompose toujours l'**Objectif** en 2 propriétés

- ▶ L'**Invariant** (Propriété 1), vrai à l'initialisation ainsi qu'à chaque itération.
- ▶ La **Condition d'arrêt** (Propriété 2) de la boucle.

On a donc le modèle d'algorithme suivant :

```
1 # initialisation des variables
2 assert invariant, "erreur initialisation"
3 while not (condition arret):
4     # traitement de la boucle et itération
5     assert invariant, "erreur iteration"
6 assert invariant et condition arret, "erreur objectif"
```

## Rappel du modèle de code

```
1 # La pré-condition  $X \geq 0$  et  $Y > 0$  doit être vérifiée
2 q = ?
3 r = ?
4 # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit être vérifié
5 while r  $\geq$  Y:
6     # Modification de q et r
7     q = ?
8     r = ?
9     # L'Invariant :  $X == q * Y + r$  et  $0 \leq r$  doit être vérifié
10 # L'Objectif :  $X == q * Y + r$  et  $0 \leq r < Y$  doit être atteint
```

- ▶ On autorise la multiplication  $*$  uniquement dans le cadre de la vérification.
- ▶ Chaque vérification est codée avec l'instruction `assert`

## Mise au point avec **assert**

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 assert X>=0 and Y>0, "erreur precondition"
4 r=?
5 q=?
6 assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while ???:
8     r=?
9     q=?
10    assert X==q*Y+r and 0<=r, "erreur iteration"
11    assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12    print ("q=",q,"r=",r)
```

On obtient :

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 assert X>=0 and Y>0, "erreur precondition"
4 r=X
5 q=0
6 assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while r>=Y:
8     r=r-Y
9     q=q+1
10    assert X==q*Y+r and 0<=r, "erreur iteration"
11    assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12    print("q=",q,"r=",r)
```

## Erreur d'initialisation

r et q sont incorrectement initialisés :

- Le programme s'arrête sur l'assert d'initialisation.

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 assert X>=0 and Y>0, "erreur precondition"
4 r=0
5 q=0
6 assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while r>=Y:
8     r=r-Y
9     q=q+1
10    assert X==q*Y+r and 0<=r, "erreur iteration"
11 assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12 print("q=",q,"r=",r)
```

```
5
3
-- AssertionError Traceback (most recent call last)
AssertionError: erreur initialisation
```



## Erreur dans l'itération

$r$  et  $q$  sont modifiés d'une façon non convenable :

- Le programme s'arrête sur l'assert d'itération.

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 assert X>=0 and Y>0, "erreur precondition"
4 r=X
5 q=0
6 assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while r>=Y:
8     r=r-Y
9     q=q+2
10    assert X==q*Y+r and 0<=r, "erreur iteration"
11 assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12 print("q=",q,"r=",r)
```

```
5
3
-- AssertionError Traceback (most recent call last)
AssertionError: erreur iteration
```

## Erreur dans l'arrêt de boucle

La condition de boucle est incorrecte :

- Le programme s'arrête sur l'assert d'objectif.

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 assert X>=0 and Y>0, "erreur precondition"
4 r=X
5 q=0
6 assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while r>Y:
8     r=r-Y
9     q=q+1
10    assert X==q*Y+r and 0<=r, "erreur iteration"
11 assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12 print("q=",q,"r=",r)
```

```
6
2
-- AssertionError Traceback (most recent call last)
AssertionError: erreur objectif
```

## Finalisation du code

Une fois le code mis au point :

- ▶ On désactive les asserts (commentaire ou suppression).
- ▶ On évite ainsi le coût d'exécution associé.

```
1 X=int(input("X="))
2 Y=int(input("Y="))
3 # assert X>=0 and Y>0, "erreur precondition"
4 r=X
5 q=0
6 # assert X==q*Y+r and 0<=r, "erreur initialisation"
7 while r>=Y:
8     r=r-Y
9     q=q+1
10    # assert X==q*Y+r and 0<=r, "erreur iteration"
11    # assert X==q*Y+r and 0<=r and r<Y, "erreur objectif"
12    print("q=",q,"r=",r)
```

- ▶ Dans le cadre de ce cours, on utilisera des fonctions de vérifications fournies (et théoriquement vérifiées). Evidemment, elles ne seront utilisées que pour la mise au point, puis à commenter.
  - ▶ exemple : robots qui ramassent des objets. On vérifie que tous les objets sont ramassés (pas efficace mais pas la solution)
  - ▶ exemple : recherche d'un maximum. On utilise un maximum sur une sous-partie...

## Exercices

# Exercice : Rédiger un assert

## Objectif donné, insérer les asserts

- Soit un programme faisant la somme des éléments d'une liste : Retrouver un invariant qui a mené à la boucle et écrivez l'assert correspondant
- Écrivez les 3 lignes (avec ou sans assert) et leur position qui permettent de tester la correction de la boucle.

```
1 liste = eval(input())
2
3 somme = 0
4 i=0
5
6 while i < len( liste ) :
7
8     somme = somme + liste[i]
9     i = i+1
10
11
12 print(somme)
```

## Exercices

### Exercice : Rédiger un assert

Invariant : `somme = sum(liste[0:i])`

#### Objectif donné, insérer les asserts

- Soit un programme faisant la somme des éléments d'une liste : Retrouver un invariant qui a mené à la boucle et écrivez l'assert correspondant
- Écrivez les 3 lignes (avec ou sans assert) et leur position qui permettent de tester la correction de la boucle.

```
1 liste = input()
2
3 somme = 0
4 i=0
5
6 while i < len(liste):
7
8     somme = somme + liste[i]
9     i = i+1
10
11 print(somme)
```

# Exercices

## Exercice : Rédiger un assert

### Objectif donné, insérer les asserts

- Soit un programme faisant la somme des éléments d'une liste : Retrouver un invariant qui a mené à la boucle et écrivez l'assert correspondant
- Ecrivez les 3 lignes (avec ou sans assert) et leur position qui permettent de tester la correction de la boucle.

```

1 liste = eval(input())
2
3 somme = 0
4 i=0
5
6 while i < len(liste):
7
8     somme = somme + liste[i]
9     i = i+1
10
11 print(somme)

```

```

1 liste = eval(input())
2 somme = 0
3 i=0
4 assert somme==sum(liste[0:i]), "erreur initialisation "
5 while i < len(liste):
6     somme = somme + liste[i]
7     i = i+1
8     assert somme==sum(liste[0:i]), "erreur iteration "
9 assert somme==sum(liste[0:i]) and i==len(liste), "erreur objectif "
10 print(somme)

```



## Exercice : indices des min, max

- ▶ Problème :
  - ▶ Calculer les indices du min et du max d'une liste L contenant au moins deux éléments.
- ▶ Exigences :
  - ▶ Les éléments de L sont distincts les uns des autres
  - ▶ Les éléments de L ne sont pas modifiés
- ▶ Le modèle de solution :
  - ▶ On parcourt cette liste au plus une fois
- ▶ Remarque : dans cet exercice, il y a un seul min et un seul max car les éléments de L sont différents entre eux.
- ▶ Les fonctions **min(L)** et **max(L)** de python seront utilisées lors de la mise au point et uniquement dans les expressions des assert.

Rappel : la notation `L[deb : long]` est la liste des éléments de l'indice `deb` à **long-1**.

- ▶ a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- ▶ b) selon un modèle de solution obtenir à partir de l'objectif :  
l'**Invariant** et la **condition d'arrêt**
- ▶ c) écrire le code du programme

## └ Exercices

### └ Exercice : indices des min, max

- ▶ a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- ▶ b) selon un modèle de solution obtenu à partir de l'objectif :  
l'invariant et la condition d'arrêt
- ▶ c) écrire le code du programme

Soit  $\text{imin}$  la variable qui stocke l'indice de minimum et  $\text{imax}$  l'indice du maximum de la liste

- Préconditions :  $\text{len}(L) > 1$  et (pour tout  $i = j : 0 \leq i, j < \text{len}(L)$ , on a  $L[i] \neq L[j]$ )!
- Modèle de solution : parcourir la liste de gauche à droite
- A une itération,
  - $\text{imin}$  et  $\text{imax}$  représentent les indices du min et du max des cases déjà traitées,
  - on traite une nouvelle case d'indice  $i$  en mettant à jour  $\text{imin}$  et  $\text{imax}$
- L'invariant proposé est donc  $\text{imin} \neq \text{imax}$  et  $0 \leq \text{imin} < i$  et  $0 \leq \text{imax} < i$  et  $i < \text{len}(L)$  et  $L[\text{imin}] = \min(L[0:i])$  et  $L[\text{imax}] = \max(L[0:i])$

## └ Exercices

### └ Exercice : indices des min, max

- ▶ a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- ▶ b) selon un modèle de solution obtenu à partir de l'objectif : l'invariant et la condition d'arrêt
- ▶ c) écrire le code du programme

- Quand s'arrête t'on? On s'arrête quand toutes les cellules sont déjà traitées
- $i = \text{len}(L)$  est la condition d'arrêt. La condition de la boucle s'écrit donc  $i \neq \text{len}(L)$
- On peut donc "découper" le code pour y ajouter les propriétés (objectif, invariant...)

## └ Exercices

## └ Exercice : indices des min, max

- ▶ a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- ▶ b) selon un modèle de solution obtenu à partir de l'objectif : l'invariant et la condition d'arrêt
- ▶ c) écrire le code du programme

```
1 L = eval(input())
2
3 # initialiser i,imin,imax
4 # verifier invariant  $0 \leq \text{imin} \leq \text{imax} < i < \text{len}(L)$  et  $L[\text{imin}] = \min(L[0:i])$  and
   L[imax]=Max(L[0:i])
5 while i != len(L):
6     #traiter la cellule i et mettre a jour imin et imax
7     # verifier invariant  $0 \leq \text{imin} \leq \text{imax} < i < \text{len}(L)$  et  $L[\text{imin}] = \min(L[0:i])$  and
       L[imax]=Max(L[0:i])
8
9 # verifier invariant  $0 \leq \text{imin} \leq \text{imax} < i < \text{len}(L)$  et  $L[\text{imin}] = \min(L)$  and
   L[imax]=Max(L)
```

## └ Exercices

## └ Exercice : indices des min, max

- a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- b) selon un modèle de solution obtenu à partir de l'objectif : l'invariant et la condition d'arrêt
- c) écrire le code du programme

```

1 L = eval(input())
2 # initialiser i,imin,imax
3
4 assert 0<=imin and imin<i and i<=len(L) and 0<=imax and imax<i and
   i<=len(L) and L[imin]=min(L[0:i]) and L[imax]=max(L[0:i]),"erreur init"
5 while i!=len(L):
6     #traiter la cellule i et mettre a jour imin et imax
7
8     assert 0<=imin and imin<i and i<=len(L) and 0<=imax and imax<i and
       i<=len(L) and L[imin]=min(L[0:i]) and L[imax]=max(L[0:i]),"erreur fin
       iteration"
9
10 assert 0<=imin and imin<len(L) and 0<= imax<len(L) and L[imin]=min(L) and
    L[imax]=max(L),"erreur objectif"

```

## └ Exercices

### └ Exercice : indices des min, max

- ▶ a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- ▶ b) selon un modèle de solution obtenu à partir de l'objectif : l'invariant et la condition d'arrêt
- ▶ c) écrire le code du programme

- D'après l'invariant,  $\text{imin}$  est l'indice du min et  $\text{imax}$  celui du max.
- Tous les éléments sont différents entre eux,  $L[\text{imin}]$  est différent de  $L[\text{imax}]$
- On va les initialiser aux valeurs des 2 premières cases pour que  $L[\text{imin}] < L[\text{imax}]$
- Et  $i$  commence donc à 2
- Cela nous donne

## └ Exercices

## └ Exercice : indices des min, max

- a) avec l'aide de l'enseignant, identifier et préciser l'objectif
- b) selon un modèle de solution obtenu à partir de l'objectif : l'invariant et la condition d'arrêt
- c) écrire le code du programme

```

1 L=eval(input())
2 if L[0]<L[1]:
3     imin,imax=0,1
4 else :
5     imin,imax=1,0
6 i=2
7 assert 0<=imin and imin<i and i<=len(L) and 0<=imax and imax<i and i<=len(L) and
    L[imin]=min(L[0:i]) and L[imax]=max(L[0:i]),"erreur init"
8 while i!=len(L):
9     if L[i]>L[imax]:
10         imax=i
11     else :
12         if L[i]<L[imin]:
13             imin=i
14     i=i+1
15     assert 0<=imin and imin<i and i<=len(L) and 0<=imax and imax<i and i<=len(L) and
        L[imin]=min(L[0:i]) and L[imax]=max(L[0:i]),"erreur fin iteration"
16
17 assert 0<=imin and imin<len(L) and 0<= imax<len(L) and L[imin]=min(L) and
    L[imax]=max(L),"erreur objectif"
18 print ("indice du min de L=",imin)
19 print ("indice du max de L=",imax)

```