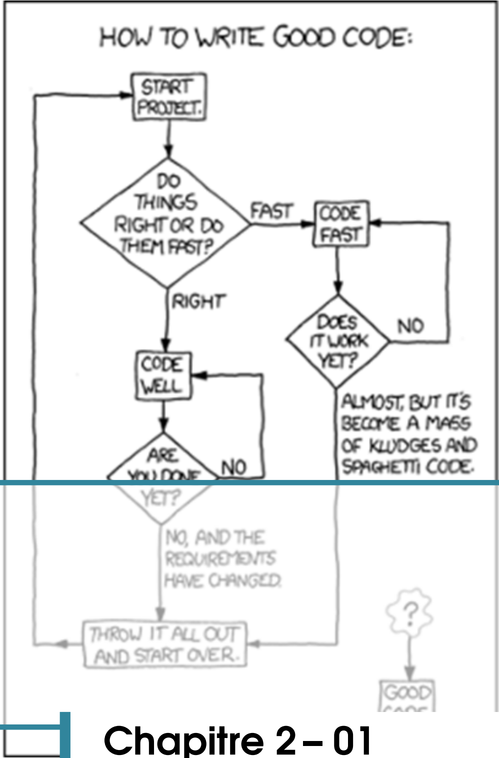

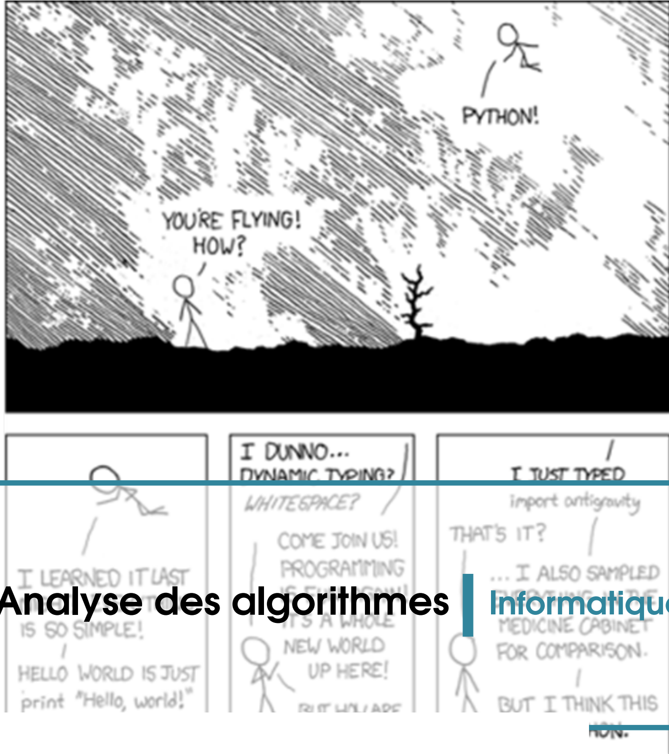

http://xkcd.com/353/
http://xkcd.com/844/

MPSI

Cycle 2

Analyse des algorithmes | Informatique

Chapitre 2 – 01
Invariance et variance

8 Janvier 2020

Cours

Savoirs et compétences :

- *AN.C1 : Justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant*
- *AN.C2 : Démontrer qu'une boucle se termine effectivement*
- *AN.S1 : Recherche dans une liste, recherche du maximum dans une liste de nombres, calcul de la moyenne et de la variance.*
- *AN.S2 : Recherche par dichotomie.*
- *AN.S4 : Recherche d'un mot dans une chaîne de caractères.*

Émilien Durif – Sylvaine Kleim
Xavier Pessoles
MPSI

1 Prouver qu'un algorithme réalise donne le résultat souhaité

1.1 Invariant de boucle

Définition Un **invariant de boucle** est une proposition vérifiée à chaque tour d'une boucle. Plus précisément, on distingue les invariants d'**entrée de boucle** et ceux de **sortie de boucle**. C'est en fait une proposition de récurrence, qui doit être vrai au début de la première boucle, puis se propager de tour de boucle en tour de boucle, jusqu'au dernier.
Les invariants de boucles ont essentiellement deux intérêts :
1. **Démontrer** l'algorithme, grâce au principe de récurrence ;
2. Répondre à notre question précédente : où commencer et où finir une boucle ?

R Il est **fortement conseillé** de toujours indiquer les invariants de boucle en commentaire dans vos algorithmes.

1.2 Preuve de correction

Définition Une preuve de correction permet de démontrer le résultat d'un algorithme. Elle repose sur l'utilisation **d'un invariant** et d'un raisonnement par **récurrence** qui possède plusieurs étapes :

Initialisation : l'invariant de boucle est vérifié au début de la première boucle ;

Supposition : que l'invariant de boucle est vérifié au début de la $n^{\text{ème}}$ boucle ;

Hérédité : à partir de la supposition on montre que l'invariant de boucle est vérifié en fin de boucle.

Conclusion : en se basant sur l'hérédité on donne l'expression de l'invariant àa la fin de la dernière boucle pour vérifier le résultat souhaité.

1.3 Exemple

■ **Exemple** Par exemple :

```
x = 5                # x = u_0 (initialisation)
for k in range(20):
    # x = u_k (invariant en entrée de boucle)
    x = 2 * x - k - 3
    # x = u_{k+1} (invariant en sortie de boucle)
# après la boucle for, k vaut 19, et x vaut donc u_{20}.
```

■

1.4 Tableau de valeurs

Définition Pour vérifier qu'un algorithme effectue bien ce qu'on lui demande, on peut utiliser un **tableau de valeurs** qui permet de déterminer en début ou fin de boucles les valeurs prises par les différentes variables.

■ **Exemple**

```
def fonctionMystere(n) :
    if n==0 or n==1:
         return 1
    else :
        res = 1
    for i in range (2,n+1) :
        res = res * i
    return res
```

Q 1 : Dresser le tableau de valeur pour les variables `i` et `res`.
Q 2 : Quel est le nom mathématique usuel donné à la fonction `fonctionMystere` ? ■

2 Prouver qu'un algorithme se termine

2.1 Variant de boucle

Comment démontrer un algorithme reposant sur une boucle indéfinie ? Pour cela, nous utilisons encore les invariants de boucle, afin de prouver qu'après la boucle, le résultat est bien celui voulu.

Mais avant même cela, il y a un point épineux : la boucle **while** va-t-elle vraiment se finir ? Il faut démontrer ce que l'on appelle la **terminaison** de l'algorithme. C'est là que réside le danger des boucles **while** : si elles sont mal écrites, la condition de la boucle ne devient jamais fausse, et la boucle est infinie.

> **Définition** Pour montrer la terminaison d'un algorithme, on utilise cette fois un **variant** de boucle. Il consiste à mettre en avant une variable dont la valeur est modifiée au cours des tours de boucle, de telle sorte que cette modification finisse par rendre fausse la condition de la boucle.

2.2 Preuve de terminaison

> **Définition** Une preuve de terminaison repose sur l'utilisation d'un variant de boucle qui est généralement associé à la condition donné à la boucle `while.` La plupart des conditions associées au boucles indéfinies sont des inégalités. Il suffit alors généralement à montrer que le variant de boucle est défini par une suite d'entiers strictement croissante ou décroissante qui ne sera donc pas bornée.

2.3 Exemple

■ **Exemple** Rechercher le premier entier n tel que la somme des entiers de 1 à n dépasse 11.

```
n = 1
s = 1          # s = somme des i de 1 àn
while s < 11 : # invariant : s = somme des i de 1 àn
    n = n + 1
    s = s + n  # variant : s, qui est entier et strictement croissant
```

■

Cycle 01

TD

Sources :

Savoirs et compétences :

- *AN.C1 : Justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant*
- *AN.C2 : Démontrer qu'une boucle se termine effectivement*
- *AN.S1 : Recherche dans une liste, recherche du maximum dans une liste de nombres, calcul de la moyenne et de la variance.*
- *AN.S2 : Recherche par dichotomie.*
- *AN.S4 : Recherche d'un mot dans une chaîne de caractères.*

3 Applications

3.1 Test de primalité

On veut tester si un entier n est premier :

```
def est_premier(n):
    """ Renvoie True si n est premier, False sinon
       Préconditon : n est un entier."""
    b = True
    for d in range(2,n):
        # b => n non divisible par 2, 3, ..., d-1.
        if n % d == 0:
            b = False
    # b <=> n premier
    return b
```

Remarque : les derniers tours de boucle sont inutiles dès que la variable b a été mise à `False`. Les exécuter tout de même est une perte de temps. Il existe plusieurs possibilités pour améliorer cela :

L'instruction `break` :

```
def est_premier(n):
    """ Renvoie True si n est premier, False sinon
        Préconditon : n est un entier."""
    b = True
    for d in range(2,n):
        # b => n pas divisible par 2, 3, ..., d-1.
        if n % d == 0:
            b = False
            break
    # b <=> n est premier
    return b
```

L'utilisation d'un `return` en milieu de boucle, à favoriser en Python pour une question de style et d'élégance :

```
def est_premier(n):
    """ Renvoie True si n est premier, False sinon
```

```
        Préconditon : n est un entier."""
    for d in range(2,n):
        # n n'est pas divisible par 2, 3, ..., d-1.
        if n % d == 0:
            return False
    return True
```

3.2 Conjecture de Syracuse

Donnons ces invariant et variant pour l'algorithme de vérification de la conjecture de syracuse (nous ne pouvons malheureusement pas le faire pour l'exemple **??**, puisque comme son nom l'indique, la conjecture de Syracuse n'a jamais été demontrée) :

Conjecture de Syracuse :
On note $f : \mathbb{N}^* \to \mathbb{N}^*$ l'application vérifiant, pour tout n pair $f(n) = n/2$ et tout n impair et $f(n) = 3n+1$.
On conjecture que pour tout entier n, il existe k tel que $f^k(n) = 1$.
Voici un algorithme calculant, pour tout n donné, le plus petit entier k vérifiant $f^k(n) = 1$:

```
def f(n):
    """Fonction de Syracuse.
    Précondition : n est un entier strictement positif"""
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1

def syracuse(n):
    """Renvoie le premier entier k tel que f\^k(n) = 0.
    Précondition : n est un entier strictement positif"""
    x = n
    k = 0
    while x != 1:
        x = f(x)
        k = k + 1
    return k
```

3.3 Fonction mystère

On considère la fonction suivante.

```
def mystere(a,b) :
    """Précondition : a,b sont des entiers, a>0, b>1"""
    k,p = 0,1
    while a % p == 0 :
        k = k+1
        p = p*b
    return k-1
```

Q 1 : Dresser un tableau de valeurs décrivant les valeurs des variables `k` et `p` en entrée des trois premiers tours de la boucle `while` de la fonction `mystere(a,b)`.
On pourra au besoin faire intervenir les variables `a` et `b`.

Q 2 : En s'aidant de la question précédente, écrire un invariant de boucle pour la boucle `while` de la fonction `mystere(a,b)`. On justifiera la réponse.

Q 3 : Donner un variant de boucle pour la boucle `while` de la fonction `mystere(a,b)`. On justifiera la réponse.

Q 4 : Déduire des questions précédentes qu'un appel de la fonction `mystere(a,b)` renvoie un résultat et déterminer le résultat alors renvoyé. On justifiera la réponse.