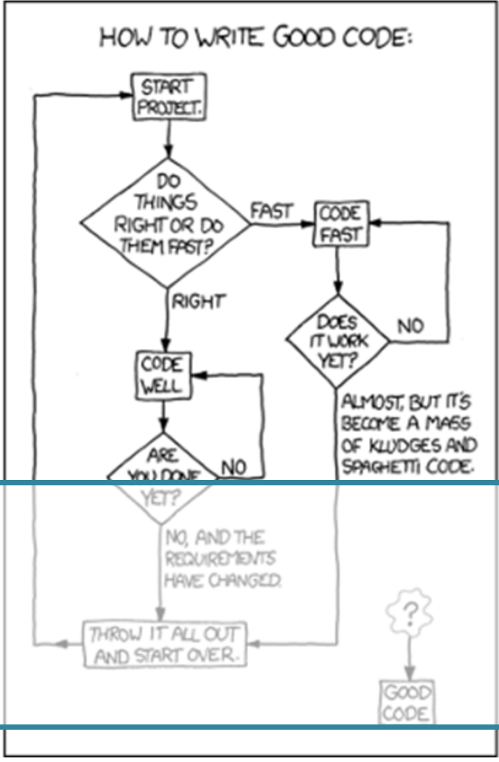
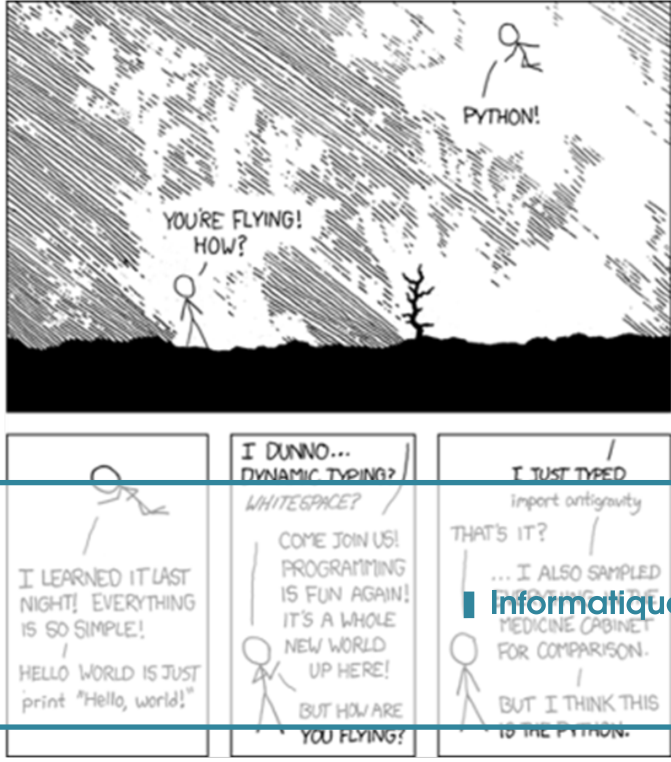
http://xkcd.com/353/

http://xkcd.com/844/





MPSI

Semestre 2

Informatique

Fiche

Introduction à la programmation en Python

Informatique

1	**Introduction**	**2**
1.1	Un premier exemple	2
1.2	Un deuxième exemple : $n!$	2
1.3	Un troisième exemple : algorithme d'Euclide	2
1.4	Un troisième exemple (bis) : algorithme d'Euclide	2
1.5	Quatrième exemple	3

E. Durif – X. Pessoles
J.-P. Berne
MPSI

1 Introduction

Définition Preuve d'algorithme

Une preuve d'algorithme est une démonstration montrant qu'un algorithme réalise la tâche pour laquelle il a été conçu.

Il faut alors montrer sa **terminaison** c'est-à-dire montrer que l'algorithme se termine. On utilise pour cela un **variant de boucle**.

Il faut ensuite montrer sa **correction** c'est-à-dire montrer que l'algorithme réalise la tâche attendue. On utilise pour cela un **invariant de boucle**.

1.1 Un premier exemple

Donner l'algorithme permettant de déterminer le plus petit entier n tel que $1+2+\ldots+n$ dépasse strictement 1000.
Proposons cet algorithme.

```
res = 0
n = 0
while res < 1000 :
    n = n+1
    res = res+n

print(n,res)
```

1.2 Un deuxième exemple : $n!$

```
for i in range(1,n+1):
    # en entrant dans le ième tour de boucle, p = (i-1)!
    p=p*i
    # en sortant du ième tour de boucle, p = i!

print(p) #p = n!
```

Ici, l'invariant de boucle est « p contient $(i-1)!$ » :

1. c'est bien une propriété qui est vraie pour $i=1$;
2. supposons qu'au rang i, $p=(i-1)!$ à l'entrée de la boucle. Au cours de la boucle, p va prendre la valeur $p=(i-1)!\times i=i!=((i+1)-1)!$ donc la propriété est vérifiée en sortie de boucle ;
3. enfin, au dernier tour de boucle, i vaut n donc $p=n!$ ce qui répond à la question.

1.3 Un troisième exemple : algorithme d'Euclide

`https://lgarcin.github.io/CoursPythonCPGE/preuve.html`

```
def pgcd(a, b):
  while b!= 0:
      a, b = b, a % b
  return a
```

On suppose que l'argument b est un entier naturel. En notant b_k la valeur de b à la fin de la $k^{\text{ème}}$ itération (b_0 désigne la valeur de b avant d'entrer dans la boucle), on a $0\leq b_{k+1}<b_k$ si $b_k>0$. La suite (b_k) est donc une suite strictement décroissante d'entiers naturels : elle est finie et la boucle se termine.

On note a_k et b_k les valeurs de a et b à la fin de la $k^{\text{ème}}$ itération (a_0 et b_0 désignent les valeurs de a et b avant d'entrer dans la boucle). Or, si $a=bq+r$, il est clair que tout diviseur commun de a et b est un diviseur commun de b et r et réciproquement. Notamment, $a\wedge b=b\wedge r$. Ceci prouve que $a_k\wedge b_k=a_{k+1}\wedge b_{k+1}$. La quantité $a_k\wedge b_k$ est donc bien un invariant de boucle. En particulier, à la fin de la dernière itération (numérotée N), $b_N=0$ de sorte que $a_0\wedge b_0=a_N\wedge b_N=a_N\wedge 0=a_N$. La fonction `pgcd` renvoie donc bien le `pgcd` de a et b.

1.4 Un troisième exemple (bis) : algorithme d'Euclide

`https://mathematice.fr/fichiers/cpge/infoprepaC8.pdf`

On effectue la division euclidienne de a par b où $a$et b sont deux entiers strictement positifs. Il s'agit donc de déterminer deux entiers q et r tels que $a=bq+r$ avec $0\leq r<b$. Voici un algorithme déterminant q et r :

```
q = 0
r = a
while r >= b :
    q = q + 1
    r = r -b
```

On choisit comme invariant de boucle la propriété $a = bq + r$.

- Initialisation : q est initialisé à 0 et r à a, donc la propriété $a = bq + r = b \cdot 0 + a$ est vérifiée avant le premier passage dans la boucle.
- Hérédité : avant une itération arbitraire, supposons que l'on ait $a = bq + r$ et montrons que cette propriété est encore vraie après cette itération. Soient q' la valeur de q à la fin de l'itération et r' la valeur de r à la fin de l'itération. Nous devons montrer que $a = bq' + r'$. On a $q' = q+1$ et $r' = r - b$, alors $bq' + r' = b(q+1) + (r-b) = bq + r = a$. La propriété est bien conservée.

Terminaison Nous reprenons l'exemple précédent.

- Commençons par montrer que le programme s'arrête : la suite formée par les valeurs de r au cours des itérations est une suite d'entiers strictement décroissante : r étant initialisé à a, si $a \geq b$ alors la valeur de r sera strictement inférieure à celle de b en un maximum de $a - b$ étapes.
- Ensuite, si le programme s'arrête, c'est que la condition du "tant que" n'est plus satisfaite, donc que $r < b$. Il reste à montrer que $r \geq 0$. Comme r est diminué de b à chaque itération, si $r < 0$, alors à l'itération précédente la valeur de r était $r' = r + b$; or $r' < b$ puisque $r < 0$. Et donc la boucle se serait arrêtée à l'itération précédente, ce qui est absurde; on on déduit que $r \geq 0$.

En conclusion, le programme se termine avec $0 \leq r < b$ et la propriété $a = bq + r$ est vérifiée à chaque itération; ceci prouve que l'algorithme effectue bien la division euclidienne de a par b.

Cycle 01

1.5 Quatrième exemple

L'objectif est de calculer le produit de deux nombres entiers positifs a et b sans utiliser de multiplication.

```
p = 0
m = 0
while m < a :
    m = m + 1
    p = p + b
```

Comme dans l'exemple précédent, le programme se termine car la suite des valeurs de m est une suite d'entiers consécutifs strictement croissante, et atteint la valeur a en a étapes.

Un invariant de boucle est ici : $p = m.b$.

- Initialisation : avant le premier passage dans la boucle, $p = 0$ et $m = 0$, donc $p = mb$.
- Hérédité : supposons que $p = mb$ avant une itération; les valeurs de p et m après l'itération sont $p' = p + b$ et $m' = m + 1$. Or $p' = (p + b) = m.b + b = (m+1)b = m'b$. Donc la propriété reste vraie.
- Conclusion : à la sortie de la boucle $p = m.b$.

Puisqu'à la sortie de la boucle $m = a$, on a bien $p = ab$.