Écrire du joli code

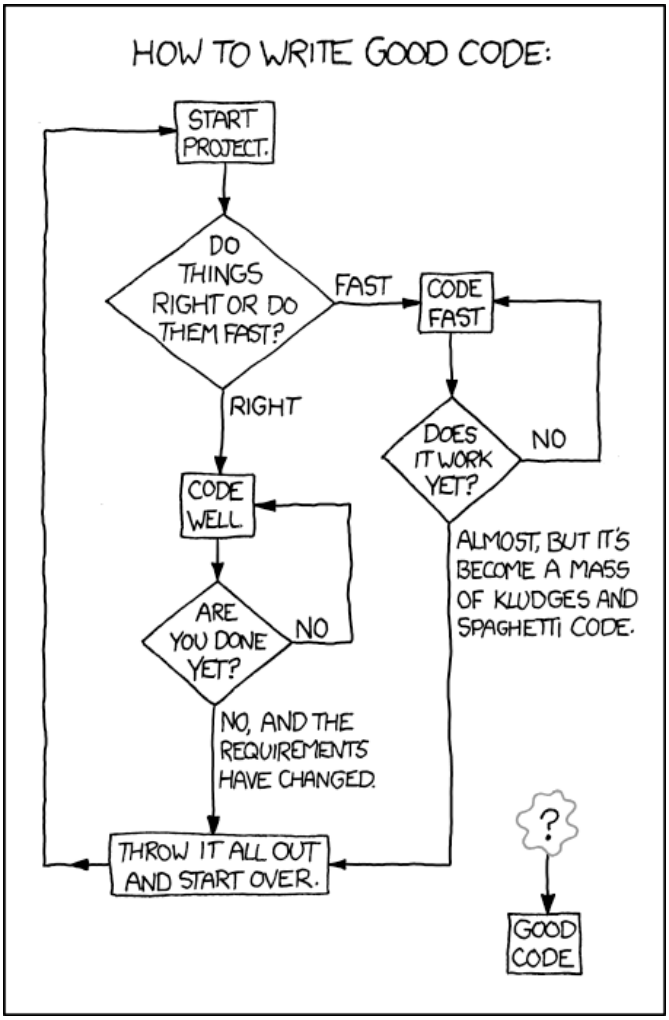
Laurent Signac – CC-BY-SA – 06-07-22 2346 ac4468d4376b62cca629

> Only ugly languages become popular. Python is the one exception.
> *Donald Knuth*

La beauté est subjective, alors, donnons quelques précisions. Du joli code est du code :

- facile à vérifier
- concis
- intelligent et efficace

C'est avéré, un code est très souvent lu par un humain : celui qui l'a écrit, celui qui reprend un projet, celui qui veut vérifier le travail d'une autre personne… Du joli code est donc du code qui facilite la relecture. La facilité de vérification (et la non redondance, mais ça va finalement ensemble) est probablement le point le plus important.



xkcd 882

1 Préférer une boucle `for` à une boucle `while`

Une boucle `while`, bien qu'elle facilite les éventuelles preuves et la manipulation d'invariants, est plus sujette à des erreurs d'inattention : évaluation incorrecte de la condition, oubli de l'initialisation, oubli de l'incrémentation. Une boucle `for`, en Python, a un côté automatique, et cette alternative est préférable lorsqu'on parcourt une collection. **Lorsque vous pouvez, utilisez de préférence une boucle `for`.**

2 Manipuler des séquences plutôt que de scalaires

Pour la manipulation des composantes d'une image, on lit souvent ceci :

```
pix = image.getpixel((x, y))
pix[0] = pix[0] // 2
pix[1] = pix[1] // 2
pix[2] = pix[2] // 2
image.putpixel((x, y), (pix[0], pix[1], pix[2]))
```

Ce qui correspond malheureusement au moins bon des deux mondes. On préférera donc une de ces deux écritures :

```
r, v, b = image.getpixel((x, y))
r, v, b = r // 2, v // 2, b // 2
image.putpixel((x, y), (r, v, b))
```

```
pix = image.getpixel((x, y))
pix[0] = pix[0] // 2
pix[1] = pix[1] // 2
pix[2] = pix[2] // 2
image.putpixel((x, y), pix)
```

3 Don't repeat yourself (dry) : Écrire une fonction

Si vous avez des portions de code qui se ressemblent et qui apparaissent plusieurs fois, même si le code fonctionne correctement, récrivez le avec un fonction.

Par exemple, si on souhaite écrire du code qui sort la liste des diviseurs stricts communs à deux nombres, on pourrait écrire :

```
ens1 = set()
for k in range(1, n1):
    if n1 % k == 0:
        ens1.add(k)
for k in range(1, n2):
    if n2 % k == 0:
        ens2.add(k)
print(ens1 & ens2)
```

Mais il vaut mieux faire ainsi :

```
def ens_div(n):
    ens = set()
    for k in range(1, n):
        ens.add(n)
    return ens

print(ens_div(n1) & ens_div(n2))
```

Ainsi, le code pour la recherche des diviseurs n'apparaît qu'une fois. Si on trouve une solution plus efficace pour tous les trouver (il y en a…), on a une seule portion de code à modifier. En outre, la seconde version est plus facile à comprendre, et elle est moins sensible aux erreurs d'inattention de copier/coller.

3.1 Don't repeat yourself (dry) : Utiliser une liste

Ce n'est pas une règle absolue, mais si vous avez besoin d'une variable `a1`, d'une variable `a2`, `a3`, `a4` et `a5`… probablement que c'est d'une liste dont vous avez besoin.

Ainsi ce code :

```
import random

a1 = random.randint(1, 6)
a2 = random.randint(1, 6)
a3 = random.randint(1, 6)
a4 = random.randint(1, 6)
a5 = random.randint(1, 6)

s = (a1 + a2 + a3 + a4 + a5) / 5
print("En moyenne avec 5 lancers de dés : ", s)
```

pourrait avantageusement être remplacé par :

```
import random

lst = []
for i in range(5):
    lst.append(random.randint(1, 6))

s = sum(lst) / 5
print("En moyenne avec 5 lancers de dés : ", s)
```

Ici, on pourrait aussi écrire du code générique pour `n` nombres, ce serait encore mieux.

Si vous avez quelques valeurs à traiter, posez-vous la question : « S'il y avait plus de valeurs, comment faudrait-il procéder ? ». Puis faites comme ça.