Les bonnes pratiques de la programmation

19 Novembre 2018

Pourquoi ?

- Eviter les bugs
- Relire du code
- Réutiliser du code
- Partager des tâches
- Gagner du temps

On lit plus de code qu’on en écrit !

Et comme pour les bons romans, du code peut être plus ou moins bien écrit !

De manière générale, il faut:

- Des commentaires, pertinents, sans excès !
- Du code aéré, suivant les conventions (en python, PEP8)
- Bien nommer ses variables et fonctions
- Documenter de manière générale son code, et interfaces

Un exemple…

Devinez-vous de quelle fonction il s’agit ?

- Perte de temps…
- Casse-tête !

```
def f(p):
    z=1
    u=p
    while u>z:
        u=u-1
        p=p*u
    return p
```

Avoir une interface

La partie la plus importante d'une fonction est son entête

- Définition claire
- Compréhension rapide
- Utilisation et test facile

```
def factorielle(n):
    """
    Calcule n! == 1 * 2 * 3 * ... * n
    """
    z = 1
    u = n

    while u > z:
        u = u - 1
        p = p * u

    return n
```

Un code clair

Rendre son code plus lisible

- Noms de variable
- Commentaires pour les parties complexes
- Pas de commentaires inutiles

```
def factorielle(n):
    """
    Calcule n! == 1 * 2 * 3 * ... * n
    """
    resultat = 1
    i = 1

    # Multiplie resultat par chaque i <= n
    while i <= n:
        resultat *= i
        i += 1

    return resultat
```

Variez l'implémentation (sans abuser) !

```python
def factorielle(n):
    """
    Calcule n! == 1 * 2 * 3 * ... * n
    """
    resultat = 1

    for i in range(2, n + 1):
        resultat *= i

    return resultat
```

```python
def factorielle(n):
    """
    Calcule n! == 1 * 2 * 3 * ... * n
    """
    return 1 if n == 1 else n * factorielle(n - 1)
```

Factoriser son code

Comme en maths !

- On peut simplifier du code
- On peut factoriser du code (boucles, fonctions) et éviter les répétitions
- Réutiliser du code existant
- Le code devient plus lisible
- Et donc plus juste et facilement modifiable et testable

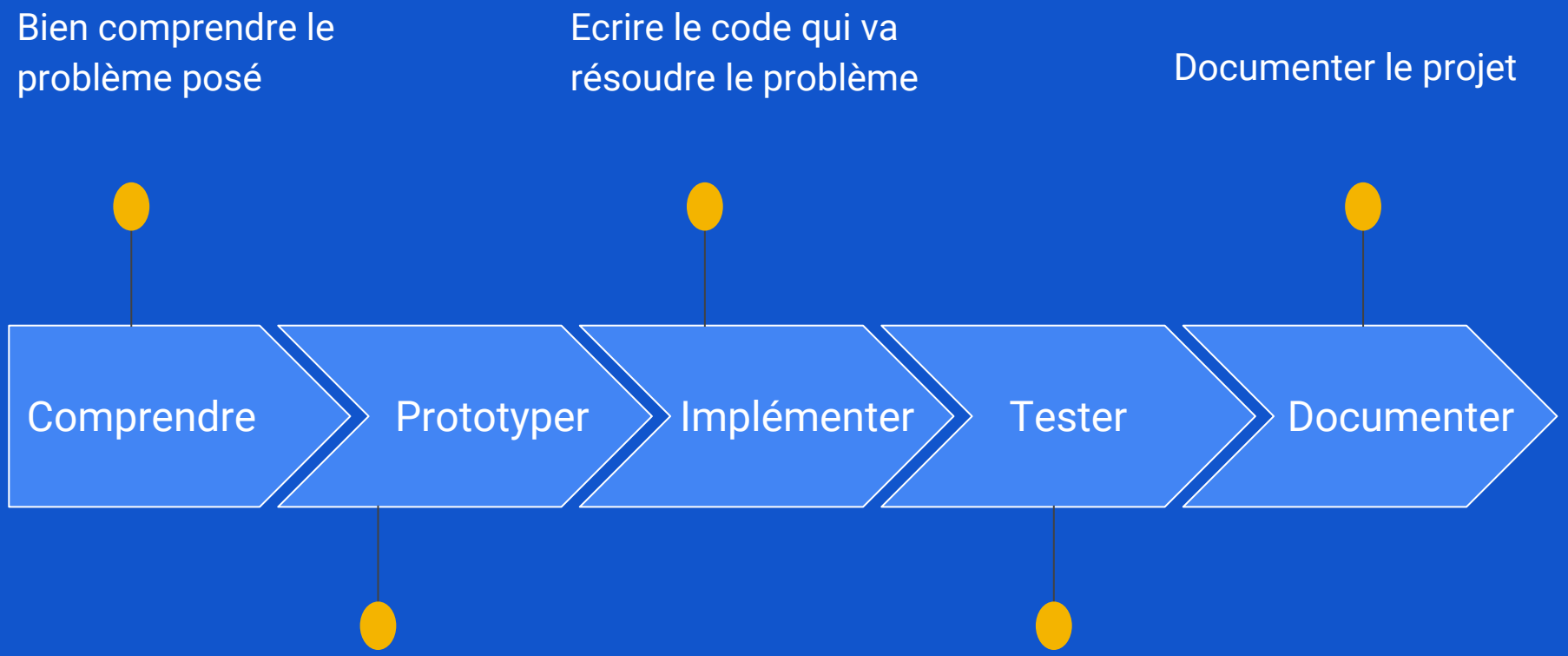
Tester son code

```
assert factorielle(1) == 1
assert factorielle(10) == 3628800
```

Tout code contient des erreurs… Pour les éviter, il faut déjà les repérer!

- Ecrire des tests pour vos fonctions
- Tester les cas limites, souvent les moins bien gérés
- Un code factorisé est plus facilement testable !

CYCLE DE DÉVELOPPEMENT

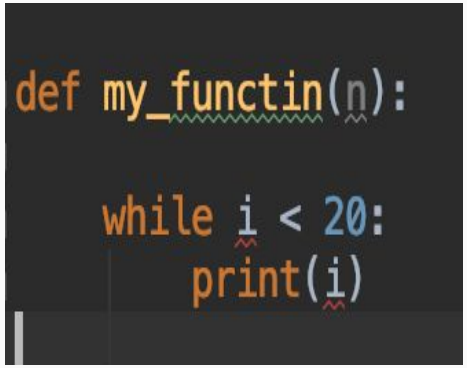


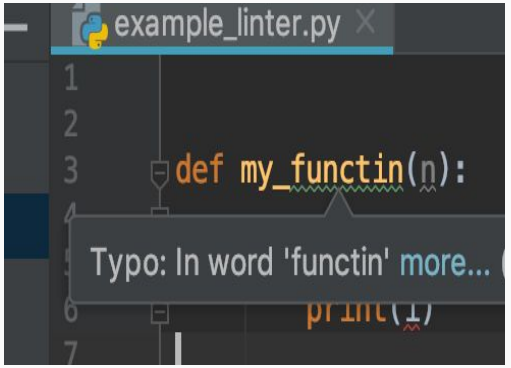
S’aider de tous les outils à disposition !

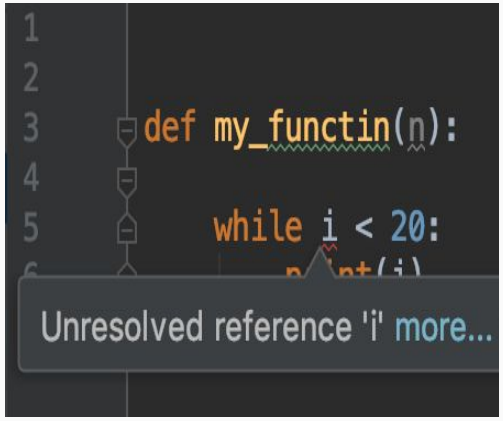
Choisir un bon environnement

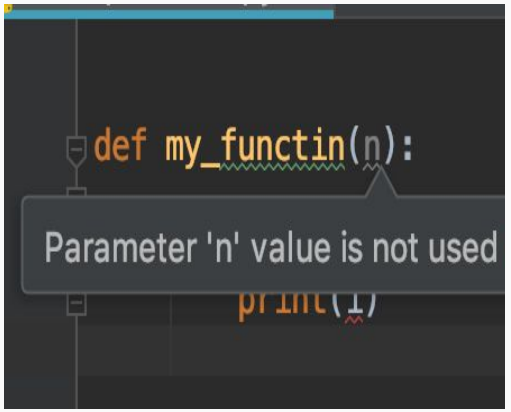
Un bon éditeur vous permettra de détecter des problèmes syntaxique ou des erreurs flagrantes au moment ou vous les écrivez

- PyCharm
- Visual Studio Code
- IDLE (avec *pylint*)










```
$ pylint example_linter.py
example_linter.py:3:0: C0111: Missing function docstring (missing-docstring)
example_linter.py:5:10: E0602: Undefined variable 'i' (undefined-variable)
example_linter.py:6:14: E0602: Undefined variable 'i' (undefined-variable)
example_linter.py:3:15: W0613: Unused argument 'n' (unused-argument)
-----------------------------------
Your code has been rated at -36.67/10  :-(
```

Lire la documentation en ligne!

Des tonnes d'informations sont disponibles gratuitement en ligne!

Habituez-vous à **lire la documentation** de ce que vous utilisez

- Doc python 3: https://docs.python.org/fr/3/
- (Presque) toutes les questions possibles sont sur Stack Overflow: https://stackoverflow.com/

Ne pas réinventer la roue...

Si vous vous lancez dans un projet complexe, il est fort possible que vous ayez besoin de nombreuses fonctionnalités: interface graphique, communication réseau, base de donnée… Vous n'allez pas tout écrire vous même!

Les sources de librairies python:

- PyPi (disponible directement dans Anaconda): https://pypi.org/
- Github: https://github.com

Partagez votre effort !

Vous avez:

- Ecrit quelque chose dont vous êtes fiers ? PARTAGEZ !
- Trouvé un bug dans une librairie? SIGNALEZ !
- Amélioré une librairie existante ? PARTAGEZ !
- Une question insoluble? CHERCHEZ !
- Une question vraiment insoluble? DEMANDEZ !

And do all that...
in english!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.

Questions ?