LA DEMARCHE DE PROJET EN NSI

La démarche de projet est une succession d'étapes permettant à chaque élève de mener à terme les tâches qu'il prend en charge. Dans la mesure où il n'y parvient pas, il doit réussir à identifier précisément l'origine du problème rencontré afin que lui-même ou d'autres puisse(nt) donner des pistes de résolution ou une solution.
In fine, il s'agit de contrôler, à chaque étape, que la tâche réalisée fonctionne, grâce à des outils de contrôle que l'on appelle *recette*.

1/ Différentes étapes du projet

1.1/ Le Cahier des Charges

Document par lequel le demandeur exprime ses besoins en termes de fonctionnalités et de contraintes.

1.2/ Réalisation de l'algorithme correspondant au CdC.

Vous devez réaliser un algorithme qui réponde au CdC.

1.3/ Test de l'algorithme

Il est souvent plus judicieux de tester l'algorithme par morceaux (quand cela est possible). Il s'agit là de décomposer le problème en plusieurs sous-problèmes.
Prenons comme exemple le test de fonctionnement d'une boucle. Pour vérifier son bon fonctionnement, on fournit plusieurs valeurs en entrée. Cela permet de déterminer si la boucle se termine et si les passages dans les différents cas sont cohérents.
De la même façon, on peut tester le fonctionnement d'une fonction en vérifiant que les valeurs données en paramètre fournissent le résultat escompté.
Le résultat de ces différents tests peut se représenter sous la forme d'un tableau : on appelle cela un ***test unitaire***.

Exemple : test de la boucle. Soit le programme suivant :

```
x=1.0
while x !=0.0 :
      x=x-0.1
```

X avant la boucle	1	0.9	0.8	0.7	0.5	0.3	0.2	0.1	0
X après la boucle	0.9	0.8	0.7	0.6	0.4	0.2	0.1	0	-0.1
Comportement attendu Arrêt/ cont(:continu)	cont	cont	cont	cont	cont	cont	cont	arrêt	
Comportement obtenu Arrêt/ cont(:continu)	cont	cont	cont	cont	cont	cont	cont	Cont : Problème	Cont : Problème

Avec ce test unitaire, on s'aperçoit que cela ne fonctionne pas. Il faut donc être capable de comprendre et de trouver pourquoi cela ne fonctionne pas.

Pour s'aider, on peut effectuer le test en affichant la valeur de la variable dans la boucle pour s'assurer de la valeur de la variable :

X attendu	X obtenu	concordance
1	1	OK
0.9	0.9	OK
0.8	0.8	OK
0.7	0.7000000000000001	Pb
0.6	0.6000000000000001	Pb
0.5	0.5000000000000001	Pb
0.2	différente	Pb
0.1	différente	Pb
0	1.3877787807814457e-16	Le test ne peu pas se faire car les valeurs sont différentes

On a identifié le problème : la variable x n'atteint pas le résultat voulu (à savoir x=0.0).

Maintenant il faut trouver pourquoi.

Ici le nombre 0,1 n'est pas un nombre fini en binaire (*voir cours sur le codage des nombres décimaux*). Donc, le programme ne peut pas se terminer correctement, nous sommes en présence d'une boucle infinie.

Les solutions possibles sont : il faut modifier le test ou modifier le calcul.

1.4/ Écriture du programme à partir de l'algorithme

Une fois l'algorithme établi, il faut le traduire en programme. Cela peut se généralement se présenter de 2 façons : soit la traduction se fait ligne à ligne soit un ensemble de lignes de l'algorithme correspond à une seule instruction (appel de fonction par exemple).

1.5/ Écriture du programme sous le langage choisi et test du fonctionnement

A partir de ce moment là on peut commencer à écrire le programme dans le langage choisi (en NSI, Python).
Mais, attention, ***il est important de l'écrire morceaux par morceaux et de le tester au fur et à mesure***. Par exemple, on écrit une fonction, on teste immédiatement son fonctionnement par l'affichage de la valeur d'une variable de sortie, d'un tableau de résultat ou par l'envoi d'un message (`print`) permettant de monter que le code a fonctionné. **Il s'agit là encore d'un test unitaire.**

Une fois que ce morceau de code est bon, on peut l'inclure dans un fichier Python général, dit « programme principal ». En effet, chaque membre de l'équipe intégrera son code dans ce programme principal. Ainsi, au fur et à mesure que l'on ajoute du code à ce programme principal, il faut aussi le tester afin de vérifier que les ajouts n'empêchent pas le programme principal de fonctionner, dans sa globalité.

Exemple : Le code du programme principal est le suivant :

```
x=2
print(x)
```

Il faut ajouter le bout de programme ci-dessous. On le vérifie d'abord seul, que le résultat obtenu sur la console est le bon.

```
print('Hello World!')
```

Résultat obtenu sur la console

```
>> Hello World!
```

Le code fonctionne, on peut donc le rajouter au code précédent et vérifier le bon fonctionnement général :

```
x=2
print(x)
print('Hello World!')
```

Résultat obtenu sur la console

```
 >> 2
>>Hello World!
```

Le code rassemblé fonctionne on peut passer à la suite.

1.6/ La recette

Elle consiste à tester le programme principal et à vérifier son bon fonctionnement. A nouveau, cela peut se faire en utilisant un tableau qui montre que les variables prennent les bonnes valeurs et que les résultats et intermédiaires et finaux sont cohérents avec le CdC.
On dit qu'on « livre » le produit. Si celui-ci est bon, le client est donc servi, satisfait. On parle alors de recette.