Pourquoi le choix de Python comme langage de programmation pour la nouvelle spécialité « Numérique et Sciences Informatiques » est un mauvais choix

I. Les faits

Le programme de la spécialité NSI (Numérique et Sciences Informatiques) a été publié dans le Bulletin Officiel du 20 janvier 2019. Dans ce programme, on trouve le paragraphe suivant :

« Un langage de programmation est nécessaire pour l'écriture des programmes : un langage simple d'usage, interprété, concis, libre et gratuit, multiplateforme, largement répandu, riche de bibliothèques adaptées et bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif est à privilégier. **Au moment de la conception de ce programme, le langage choisi est Python version 3** (ou supérieure). L'expertise dans tel ou tel langage de programmation n'est cependant pas un objectif de formation. »

Malgré la dernière phrase mettant un certain bémol, le texte mis en gras semble montrer qu'il n'y a guère le choix du langage. D'ailleurs, il semble relativement logique que le langage soit le même partout en France, pour éviter un certain nombre de problèmes, notamment ceux liés au déménagement des élèves ou au changements de professeurs.

La situation est donc claire : la programmation sera effectuée en Python.

Mais pourquoi un tel langage ?

Le texte du B.O rappelé ci-dessus cite les critères qu'ils ont pris en considération :

- simple d'usage
- interprété
- concis
- libre et gratuit
- multi plate-forme
- largement répandu
- riche de bibliothèques adaptées
- bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif

Nous allons maintenant dans un premier temps examiner la pertinence de ces différents points et l'adéquation de Python avec ces critères.

II. Pertinence des critères retenus

1. Un langage simple d'usage

Choisir pour premier langage un langage simple d'usage est effectivement une considération raisonnable. Ainsi, j'ai beau apprécié beaucoup le langage Java, un langage pour lequel un simple « Hello World » nécessite d'écrire tout le code suivante :

```
public class Hello {
     public static void main(String… args) {
          System.out.println("Hello World") ;
     }
}
```

N'est pas forcément idéal, surtout quand la même chose en Python s'écrit :

```
print("Hello World")
```

2. Un langage interprété

Utiliser un langage interprété peut paraître plus simple qu'utiliser un langage compilé dans la mesure où la phase de compilation est, justement…, supprimée. Pourtant, utiliser un langage compilé présente un intérêt remarquable : l'analyse syntaxique effectuée sur tout le code dès la phase de compilation, tandis que, dans le cas d'un langage interprété, les erreurs syntaxiques ne sont détectées que tardivement, à l'exécution de la ligne de code posant problème. Ainsi, le choix d'un langage interprété, qui peut sembler plus simple au premier abord, surtout dans un monde idyllique où tout le monde écrirait des programmes directement corrects, est particulièrement mauvais, surtout pour des jeunes découvrant l'informatique et ayant besoin d'une validation forte au préalable. Certes, les I.D.E.[1] permettent souvent de détecter l'essentiel des erreurs syntaxiques, mais on quitte alors le contexte de « simplicité d'usage » placé en premier postulat.

3. Un langage concis

Si l'extrême verbosité d'un langage peut être critiquée, j'ai du mal à considérer la concision comme un critère de sélection prépondérant. Nous avons quitté l'époque du ZX-81 avec 1Ko de mémoire et des écrans 40 colonnes, et la concision est rarement un critère de qualité. Par ailleurs, une concision forte est souvent synonyme d'implicite fort, et cela nuit à la bonne compréhension d'un langage, surtout quand il s'agit de découvrir les bases de l'informatique. Les choix faits par les concepteurs de Kotlin, par exemple vont dans ce sens : concision certes, mais suppression d'énormément d'implicites par l'ajout de mots-clés. Ce n'est pas ce que l'on trouve dans Python. Un exemple typique est lié à la gestion cachée des espaces de nom, et le sens différent donné à une variable utilisée dans une fonction, suivant le fait qu'une valeur lui est affectée ou pas.

4. Un langage libre et gratuit

Il s'agit effectivement de critères fondamentaux à prendre en compte. Il est d'ailleurs surprenant que l'Éducation Nationale n'ait toujours pas imposé dans les collèges et les lycées l'utilisation de systèmes d'exploitation libres et gratuits.

1 I.D.E. : Integrated Development Environment

5. Un langage Multi plate-forme

Je pense que c'est un critère qui a plus été intégré pour faire face au fait que les lycées n'utilisent pas tous le même système d'exploitation, et que beaucoup de lycéens sont sous Windows, car la plupart des langages de nos jours sont multi-plate-formes, et certains ont un spectre d'utilisation plus large que Python, si on intègre les plate-formes mobiles.

6. Un langage largement répandu

À défaut d'être stupide, ce n'est pas forcément un critère fondamental. Ceci est d'autant plus vrai que l'informatique évolue à une telle vitesse que nul ne peut prédire si un langage « largement répandu » aujourd'hui le sera encore dans 2 ans.

7. Un langage riche de bibliothèques adaptées

Ce critère est certainement l'un des plus ridicules qui puisse être donné. Des bibliothèques adaptées ? Adaptées à quoi ? Oui, Python est riche en bibliothèques, et c'est d'ailleurs l'une des raisons de son succès. Mais il n'est pas le seul. Par ailleurs, l'utilisation d'un bibliothèque associée à un domaine particulier nécessite de bien connaître le domaine en question, et c'est rarement le cas. Surtout pour des lycéens. En 2 ans, très peu de bibliothèques pourront être utilisées intelligemment. Autrement dit, pour des lycéens, la richesse des bibliothèques n'est certainement pas un critère à prendre en compte.

8. Un langage bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif

La décision sous-jacente est claire : on se place *de facto* dans une philosphie *legacy*[2] : plutôt que de concevoir quelque chose de propre dès le départ, on se raccroche au branche, quitte à récupérer toutes les casseroles héritées du passé. Il est vrai que le CAPES d'Informatique n'existe pas encore et qu'en attendant, cette spécialité sera dispensée par des enseignants n'ayant pas reçue une formation adéquate[3].

III. Quelques critères oubliés

A. « Sûreté » du langage

Un des gros problème du développement de logiciel est la facilité avec laquelle des bugs peuvent être introduits. Utiliser dès le départ un langage fortement contraignant permet de prendre dès le départ de bonnes habitudes et s'avère donc essentiel. Et à ce niveau-là, Python est largement en-dessous du niveau :

1. Les variables ne sont pas déclarées

Il est donc facile de se tromper de nom sans que cela ne génère d'erreur.

Exemple :

```
variable = 2
variabIe = variable +1
variable = variable*2
print(variable)
```

2 Utilisation volontaire de l'anglicisme *legacy* (héritage), par allusion à l'expression très connue des informaticiens : « *code legacy* », faisant référence aux développement dans lequel on est obligé de composer avec des technologies dépassées car on n'ose pas toucher à ce qui marche.

3 L'organisation de DIU « Enseigner l'Informatique au Lycée » à destination des enseignants déjà en poste n'est qu'un leurre, dans la mesure où l'on fait croire que l'on va, en 125 heures, former des gens à un niveau « BAC+4 ».

Quelle est la valeur affichée par ce programme ? Réponse en note de bas de page[4].

2. Les variables ne sont pas typées

Une même variable peut contenir des données de différents types à différents moments. Il n'est pas non plus possible de vérifier statiquement qu'une variable est utilisée « légalement ».

Exemple :

```
a = 2
b = a * 2
print(b)
a = "2"
b = a * 2
print(b)
```

Qu'affiche ce programme ? Voir en bas de page pour la réponse[5].

3. Encapsulation

Un des concepts clés des langages objets est l'encapsulation. Cela permet d'isoler l'utilisation d'une classe via les méthodes qu'elle propose de sa structure interne. Mais cela impose de pouvoir masquer la structure de la classe. Cela n'est malheureusement pas possible en Python : toutes les méthodes et tous les attributs d'une classe sont visibles pour tout le monde.

B. Efficacité du langage

1. Efficacité écologique

On ne peut plus nier le réchauffement climatique. Il est donc indispensable d'économiser les dépenses énergétiques partout où cela est possible. Or l'influence du langage de programmation est loin d'être négligeable. Une étude[6] a été réalisée en 2018 et la place de Python sur ce plan-là est catastrophique : 75 fois plus énergivore que le C, 40 fois plus que Java, 25 fois plus que Haskell.

2. Efficacité en temps

Mise à part pour des applications très particulières, l'efficacité en temps d'un langage n'est pas un facteur primordiale. On peut cependant remarque qu'à ce niveau-là, Python est, là encore, très mal placé. La même étude que celle citée précédemment montre que Python est plus de 70 fois plus lent que le C, 38 fois plus lent que Java, et même 20 fois plus lent que Haskell.

4 Ce programme affiche la valeur 4. En effet, à la deuxième ligne, on a, par erreur, mis un « I » majuscule à la la place d'un « l » minuscule dans le nom de la variable à gauche du signe égal.

5 Ce programme affiche « 4 » puis « 22 »

6 https://thenewstack.io/which-programming-languages-use-the-least-electricity/

IV. Les raisons d’un si mauvais choix.

Le choix d’un langage pertinent comme « premier langage » n’est pas un choix facile. Pour être fait correctement, il faut avoir un certain nombre de compétences :

- une longue expérience dans l’enseignement de l’informatique, avec différents langages, notamment pour des jeunes sortant du bac
- une bonne connaissance de l’éventail des langages relativement bien répandus
- une bonne connaissance du développement de projets d’au moins 10 000 lignes de code, afin d’être sensibilisé à tous les problèmes qui peuvent apparaître sur des projets qui ne se réduisent pas à quelques centaines de lignes.

Réunir ces 3 compétences est assez rare. Ce ne sont pas les membres du Conseil Supérieur des Programmes (CSP) qui les ont (3 députés, 2 sénateurs, 1 présidente, 1 vice-président et 8 personnes « qualifiées » mais aucun informaticien). Bien sûr, le CSP a fait appel à un GEPP (Groupe d’Experts Pour les Programmes), dont voici la composition :

- **Informaticiens :**
  - **Gérard BERRY**, professeur au Collège de France, chaire « Algorithmes, machines et langages », Paris (copilote du groupe)
  - **Gilles DOWEK**, chercheur à l’Institut national de recherche en informatique et en automatique (Inria) au Laboratoire de Spécification et Vérification (LSV) ; professeur attaché à l’École normale supérieure Paris-Saclay (ENS Paris-Saclay), Paris
  - **Christine FROIDEVAUX**, professeur des Universités en informatique, LRI CNRS & Université ParisSud, Paris
  - **Jean-Gabriel GANASCIA**, professeur des Universités en informatique, Sorbonne Université, Paris
  - **Françoise TORT**, maître de conférences en informatique à l’École normale supérieure Paris-Saclay (ENS Paris-Saclay), Paris
- **Professeurs du secondaire :**
  - **Maxime FOURNY**, professeur de mathématiques, lycée Paul-Émile VICTOR, Champagnole
  - **Loïc JOSSE**, professeur de sciences industrielles de l'ingénieur (SII) en informatique et sciences du Numérique (ISN) et de sciences et technologies de l'industrie et du développement durable (STI2D), lycée LOUIS-LE-GRAND, Paris
  - **Alexis LECOMTE**, professeur de mathématiques, lycée André MAUROIS, Elbeuf
  - **Charles POULMAIRE**, professeur de mathématiques, lycée VAN GOGH, Aubergenville
- **Inspecteurs généraux :**
  - **Laurent CHENO**, inspecteur général de l’Éducation nationale, groupe Mathématiques (copilote du groupe)
  - **Pascale COSTA**, inspectrice générale de l’Éducation nationale, groupe Sciences et techniques industrielles
  - **Christine GAUBERT-MACON**, inspectrice générale de l’Éducation nationale, groupe Économie-gestion

- **Christine WEILL**, inspectrice d'académie – inspectrice pédagogique régionale de mathématiques, académie de Versailles»

- **Autres :**
  - **Dominique CARDON**, directeur du laboratoire de recherche MédiaLab, Sciences Po, Paris

On constate d'emblée le faible nombre d'informaticiens dans ce groupe : 5 personnes sur un total de 14 ! Pour connaître certaines de ces personnes et leurs travaux, je pense qu'il est difficile de remettre en doute leur compétence. De plus, entre des échanges avec les uns et la connaissance des travaux des autres, je doute qu'ils aient été tous très favorables au choix de Python.

Mais il s'avère que dans ce groupe, 8 personnes sont rattachées au cycle secondaire, que ce soit comme enseignants (4) ou comme « inspecteur général » (4). Je présume donc que ces gens ont pesé lourdement pour que l'on recommande le langage le plus utilisé à ce jour au lycée, à savoir Python. Et ainsi, d'emblée, on se place dans un contexte de « *code legacy*[7] », source reconnue depuis des années de nombreux bugs et/ou surcoûts dans le développement informatique.

C'est aussi une faute pédagogique grave (la phrase qui suit est réservée aux spécialistes) : comment justifier la partie de l'enseignement consacrée à la preuve d'invariant quand le langage support ne permet pas de forcer l'encapsulation ?

7 Situation consistant à devoir « faire avec » du code développé il y a longtemps, avec des outils dépassés, mais que, pour diverses raisons, on se refuse à ré-écrire