REFERENTIEL NORMATIF du CNES

RNC

Référence: **RNC-CNES-Q-HB-80-535**

Version 1
11 Mai 2009

Manuel

REGLES POUR L'UTILISATION DU LANGAGE PYTHON

ACCORD du Bureau de Normalisation	**BN n° 54 du 16/09/09**
APPROBATION Président du CDN Alain CUQUEL	

PAGE D'ANALYSE DOCUMENTAIRE

TITRE :	Règles pour l'utilisation du langage Python		
MOTS CLES :	Langage, Script, Python, Règle, Standard, Programmation		
NORME EQUIVALENTE :	Néant		
OBSERVATIONS :	Néant		
RESUME :	Ce document présente les règles de programmation applicables à tout projet incluant des scripts développés en Python.		
SITUATION DU DOCUMENT :	Ce document fait partie de la collection des Manuels approuvés du Référentiel Normatif du CNES. Ce document est affilié à la norme RNC-ECSS-Q-80 "Assurance Qualité des Logiciels".		
NOMBRE DE PAGES :	24	**Langue :**	Français
Progiciels utilisés / version :	Word 2002		
SERVICE GESTIONNAIRE :	Inspection Générale Direction de la Fonction qualité (IGQ)		
AUTEUR(S) : Anne-Thérèse NGUYEN (DCT/AQ/SO)			
RELECTURE / CONTROLE : Jean-Charles Damery (DCT/AQ/SO)			

PAGES DES MODIFICATIONS

VERSION	DATE	PAGES MODIFIEES	OBSERVATIONS
1	11/05/2009		Suite à la FEB 91/2009 acceptée au BN n° 54 du 16/09/09, introduction dans le RNC du manuel RNC-CNES-Q-HB-80-535.

TABLE DES MATIERES

1. INTRODUCTION

Le document « Règles pour l'utilisation du langage Python » est rattaché à la norme RNC-ECSS-Q-ST-80 « Assurance qualité des logiciels ». Il décrit les règles applicables pour un script développé en utilisant le langage Python.

2. OBJET

Le but de ce manuel est d'établir les règles et recommandations pour l'utilisation du langage Python pour le développement de scripts élaborés. Ces règles et recommandations ont été établies à partir de « l'état de l'art » et du « retour d'expérience » cumulé sur les projets.

Ce document n'est pas un manuel de référence du langage Python, et ne traite pas des aspects interactifs de l'interpréteur de commandes. Son utilisation nécessite la connaissance du langage Python.

Nous attirons l'attention du lecteur sur le fait que le vocabulaire utilisé pour Python est légèrement différent des autres langages orientés objet. Nous demandons au lecteur de bien appréhender la définition des termes utilisés pour ne pas être déstabilisé lors de la lecture de ce manuel.

3. DOMAINE D'APPLICATION

Ce document est applicable au développement et à la maintenance des systèmes informatiques, pour leurs parties réalisées en Python. Il est complété par le document « Règles communes pour l'utilisation des langages de programmation » (DA1).

Pour utiliser les règles Python sur un projet, il faudra suivre la procédure suivante :

- Sélectionner dans les règles communes et les règles Python, les règles applicables au projet en fonction des critères de tailorisation ; cette sélection s'effectuera avec l'outil de tailorisation.
- Adapter certaines règles au projet.

Le document est ainsi destiné à plusieurs types de lecteurs :

- le chef de projet qui doit spécifier correctement l'application Python à développer,
- le chef de projet et/ou l'ingénieur qualité qui doit sélectionner les règles et éventuellement adapter et compléter les règles en fonction du contexte de réalisation,
- les personnes chargées de la réalisation du projet : elles doivent appliquer les règles retenues.

4. DOCUMENTS

4.1. DOCUMENTS DE REFERENCE

DR	Identification	Titre
(DR1)	RNC-ECSS-Q-ST-80	Assurance Produit des Logiciels

4.2. DOCUMENTS APPLICABLES

DA	Identification	Titre
(DA1)	RNC-CNES-Q-HB-80-501	Règles communes pour l'utilisation des langages de programmation.

4.3. AUTRES DOCUMENTS

- Au Coeur de Python version 2.5 – Wesley J.Chun - Editions CampusPress
- PEP 8 Style Guide for Python Code – Guido van Rossum - http://www.python.org/peps/pep-0008.html
- PEP 352 Required Superclass for Exceptions - Guido van Rossum, Cannon - http://www.python.org/peps/pep-0352.html
- PEP 308 Conditional Expressions – Guido van Rossum, Hettinger - http://www.python.org/peps/pep-0308.html
- PEP 3000 Python 3000 – Guido van Rossum - http://www.python.org/peps/pep-30000.html
- PEP 257 Docstring Conventions – David Goodger - http://www.python.org/peps/pep-0257.html
- PEP 287 reStructuredText Docstring Format – David Goodger - http://www.python.org/peps/pep-0287.html
- Python Style Guide – Guido van Rossum – http://www.**python**.org/doc/essays/**styleguide**.html
- Python Reference Manual – Guido van Rossum – http://docs.python.org/ref/ref.html
- Python tutorial - Guido van Rossum - http://docs.python.org/tut/tut.html
- Python library reference – Guido van Rossum - http://docs.python.org/lib/lib.html

5. TERMINOLOGIE

5.1. GLOSSAIRE

Terme	Définition
Docstring	Un docstring est une chaîne de caractères entre guillemets permettant de décrire le code. Un docstring est habituellement utilisé pour décrire un module, une classe, une fonction ou méthode, un attribut donnée, une variable.
Module	Un module python contient des blocs de code à exécuter, des déclarations de classe, de fonctions. De manière générale, est considéré comme module tout fichier contenant du code Python. Le nom du fichier est alors le même que le module complété de l'extension « .py ». Un module peut être exécuté en tant que script ou être importé dans un autre code python. A la différence d'autres langages pour lesquels il est possible d'importer des classes, en Python les imports concernent uniquement les modules.
Package	Un package réunit un ensemble de modules et de sous packages. Chaque package ou sous package contient en particulier un module __init__ en charge de son initialisation lors de l'import.
Objet	Un objet python est un élément possédant un identificateur unique (adresse mémoire), un type, une valeur. Cela concerne les nombres, les chaînes de caractères, les listes, les fonctions, les méthodes, les modules, les classes, les fichiers, etc. Si certains objets ont des attributs, on accède à ces attributs par la notation pointée.
Classe	Une classe est une forme programmée d'abstraction du monde réel. Concrètement en python, c'est une structure de données comprenant des caractéristiques comportementales. Elle joue le rôle d'un patron (modèle) à la création d'objets « réels », les instances. Une classe possède des attributs donnée et des attributs fonctionnels (méthodes).
Attribut de classe	On appelle attribut de classe, un attribut donnée ou méthode dont l'accès s'effectue sans avoir à créer une instance. Cette appellation est équivalente à la notion « static » plus communément employée en C++ et en Java.

5.2. ABREVIATIONS

OO	Orienté Objet

6. ADEQUATION EVENTUELLE AUX NORMES ET STANDARDS

Les règles présentées dans ce document sont conformes à la version 2.5 et prépare à la version 3.0 de Python.

7. LES REGLES

7.1. ORGANISATION DU CODE

Org.InstrucClasse	Les instructions d'une classe doivent être toutes contenues dans les définitions des méthodes de la classe.
M=3;F=1;P=43;V=1	
Quelconque	

Description

Une classe Python peut contenir du code hors méthode. Ce code est alors exécuté lors du chargement du module contenant la classe. Cette possibilité du langage ne doit pas être utilisée.

Justification

Cette règle améliore la lisibilité du code et permet le respect de la notion d'encapsulation des langages OO.

Exemple

L'exemple suivant est à éviter :

```
class NomClasse:
    <instruction-1>
    .
    .
    .
    <instruction-N>
```

Préférer :

```
class NomClasse :
       def fonction(self) :
          <instruction-1>
           .
           .
           .
          <instruction-N>
```

Org.DeclAttributs	Toutes les déclarations des attributs donnée d'une classe doivent être regroupées.
M=3;F=1;P=43;V=1	
Quelconque	

Description

Un attribut donnée de classe Python ne doit pas être déclaré n'importe où dans la classe. Une zone contenant toutes les déclarations doit être définie.

Justification

Le respect de cette règle améliore la lisibilité du code.

Exemple

L'exemple suivant est à éviter car il déclare les attributs donnée __coordX, __coordY, __coordZ et __phi à différents endroits :

```
class NomClasse :


        def __init__(self,x,y,z) :
            self.__coordX = x
            self.__coordY = y
            self.__coordZ = z

        def setPhi(phi) :
            self.__phi = phi
```

Préférer l'exemple suivant dans lequel la déclaration de tous les attributs donnée est réalisée dans la méthode d'initialisation :

```
class NomClasse :

        def __init__(self,x,y,z,phi) :
            self.__coordX = x
            self.__coordY = y
            self.__coordZ = z
            self.__phi = phi

        def setPhi(phi) :
            self.__phi = phi
```

Org.DeclVariables	Une variable locale d'une fonction (resp. bloc) doit être déclarée au début de la fonction (resp. bloc).
M=3;F=1;P=43;V=1	
Quelconque	

Description

Les variables locales ne doivent pas être déclarées n'importe où dans la fonction ou dans le bloc. Elles doivent être déclarées en début de fonction.

Justification

Le respect de cette règle améliore la lisibilité du code.

Exemple

L'implémentation suivante est à éviter :

```
def calculePhi(alpha) :
   . . .
   theta = 90 + alpha
   . . .
   phi = theta + . . .
   return phi
```

Préférer :

```
def calculePhi(alpha) :
   phi = 0
```

```
theta = 0
. . .
theta = 90 + alpha
. . .
phi = theta + . . .
return phi
```

Org.Elif	La forme elif doit être utilisée à la place de la forme else : if.
M=3;F=0;P=75;V=2	
Quelconque	

Description

Le langage Python n'offre pas de structure de contrôle de type `switch` comme les langages Java ou C++. L'implémentation d'un `switch` en python implique l'utilisation de nombreuses structures if / else if imbriquées. Pour coder ce type de structure de contrôle, la forme elif doit être utilisée.

Justification

L'utilisation de `elif` permet de réduire le niveau d'indentation et améliore la lisibilité du code.

Exemple

L'implémentation suivante est à éviter :

```
if xCarre < 0 :
    print "xCarre est negatif – remise à zero"
    xCarre = 0
else :
    if xCarre == 0 :
        print " xCarre est nul"
   else :
       if xCarre > 0 :
          print " xCarre est positif"
       else :
          raise ...
```

Préférer :

```
if xCarre < 0 :
    print "xCarre est negatif – remise à zero"
    xCarre = 0
elif xCarre == 0 :
    print " xCarre est nul"
elif xCarre > 0 :
    print " xCarre est positif"
else :
    raise ...
```

7.2. PRESENTATION DU CODE

Pr.Comment	Les modules, classes, méthodes, attributs et fonctions doivent être commentés par des docstring.
M=2;F=1;P=79;V=2	
Quelconque	

Description

Comme tout programme quelque soit le langage, il est nécessaire que le code soit commenté pour assurer sa maintenabilité. Le langage python offre différentes manières d'inclure des commentaires dans le code, le « # » comme la plupart des langages de scripts sous Unix et/ou les docstrings. Outre ces commentaires fonctionnels, chaque module, classe, fonction ou attribut doit être spécifiquement commenté en utilisant le mécanisme des docstring.

Justification

Les docstring sont des chaînes de caractères spéciales. Elles sont accessibles au moment de l'exécution et peuvent servir à générer automatiquement de la documentation tout comme la javadoc pour le langage Java.

Exemple

Cf Annexe.

7.3. IDENTIFICATEURS

Id.Attributs	Dans une classe, un attribut donnée ne doit pas avoir le même nom qu'un attribut méthode.
M=3;F=3;P=4;V=1	
Quelconque	

Description

Sans Objet

Justification

Si dans une classe, un attribut donnée et un attribut méthode possède le même nom, l'attribut donnée masquera l'attribut méthode qui ne sera alors jamais appelé.

Exemple

Sans Objet

Id.AttributPrive	Dans une classe, le nom d'un attribut de type privé doit commencer par deux soulignés.
M=2;F=2;P=30;V=1	
Quelconque	

Description

Sans Objet

Justification

Cette marque permet à l'interpréteur de différencier les attributs privés des autres attributs.

Exemple

Sans Objet

Id.Fichier	Un fichier contenant du code python doit avoir l'extension « .py ».
M=2;F=0;P=105;V=2	
Quelconque	

Description

Sans Objet

Justification

L'extension « .py » est celle utilisée par convention. Elle permet de différencier un code python d'un code écrit dans un autre langage.

Exemple

Sans Objet

7.4. DONNEES

Don.DeclAttribut	Tous les attributs donnée doivent être privés par défaut.
M=2;F=1;P=45;V=1	
Quelconque	

Description

En Python, les attributs des classes sont publics par défaut. Leur accès est direct dans le module ou dans un autre module qui importe ce module. Pour maîtriser l'encapsulation des données, chaque attribut donnée qui ne nécessite pas de visibilité particulière sera explicitement déclaré privé.

Justification

Cette règle améliore la maîtrise de l'encapsulation des données. D'autre part, déclarer les attributs privés permet d'éviter tout conflit avec les espaces de nom des classes dérivés : durant l'exécution, les attributs privés sont renommés pour inclure le nom de la classe.

Exemple

En python, un attribut privé possède un nom qui débute par un double blanc souligné (__).

```
# Declaration d un attribut prive d une classe
class MaClasse :
    def __init__(self) :
        __num = 0
```

Lors de l'exécution, l'accès à l'attribut ne peut s'effectuer que via le nom de l'attribut transformé en `_MaClasse__num`.

Don.VarCoherente	Les données stockées par une même variable doivent être physiquement/typologiquement cohérentes entre elles tout le long du programme.
M=3;F=2;P=80;V=0	
Quelconque	

Description

Python ne présente aucune restriction quant à la réutilisation d'une variable dans un programme pour stocker une donnée de type différent. Cette possibilité du langage ne doit pas être utilisée.

Justification

Améliore la lisibilité et la fiabilité du code.

Exemple

Ne pas écrire :

```
angle = 90
...
angle = « rectangle »
...
```

Préférerer :

```
angle = 90
...
angleNom = « rectangle »
...
```

Don.ListeCoherente	Les données stockées dans une liste doivent être physiquement/typologiquement cohérentes.
M=3;F=2;P=80;V=0	
Quelconque	

Description

Extension de la règle Don.VarCoherente aux listes.

Justification

Améliore la lisibilité et la fiabilité du code.

Exemple

Ne pas écrire :

```
angles = [0, « rectangle », 180, 360]
```

Préférez :

```
angles = [0, 90, 180, 360]
```

7.5. TRAITEMENTS

Tr.Definition	Toutes les méthodes d'une classe doivent être définies explicitement à l'intérieur de la classe.
M=3;F=2;P=45;V=0	
Quelconque	

Description

Le langage de script Python est très permissif. Il est possible notamment de ne pas définir explicitement une méthode à l'intérieur de la classe en la faisant pointer sur une fonction globale (cf exemple).

Justification

La déclaration d'une méthode en dehors du périmètre de définition de la classe comporte différents inconvénients :

- Elle dégrade la lisibilité du code
- Elle n'est pas compatible avec la notion de paquetage des langages OO.
- Elle rajoute une fonction globale et contribue à polluer l'espace de nommage en rajoutant un nouveau nom.

Exemple

L'exemple suivant n'est pas suivre :

```
# Méthode définie en dehors de la classe
# par le biais d'une fonction globale

def f1(self, x, y):
    return min(x, x+y)

class C:
    f = f1
    def g(self):
        return 'bonjour'
```

Préférer l'implémentation suivante :

```
# Méthode définie explicitement dans la classe

class C:

    def f(self, x, y):
        return min(x, x+y)

    def g(self):
        return 'bonjour'
```

7.6. GESTION DES ERREURS

Err.ExpressionExcept	On ne doit pas utiliser l'instruction « except : » sans préciser l'exception que l'on veut traiter.
M=1;F=2;P=95;V=2	
Quelconque	

Description

Python permet d'utiliser l'instruction « except : » seule. Cette possibilité ne doit pas être exploitée.

Justification

L'utilisation de cette instruction sans préciser les exceptions à traiter implique l'interception de tous les signaux y compris les signaux de contrôle provenant du système.
Cette règle permet de maîtriser le périmètre du traitement d'exception.

Exemple

Sans Objet.

Err.ExcepChaine	Une exception ne doit pas être une chaîne de caractère.
M=2;F=1;P=95;V=2	
Quelconque	

Description

L'instruction « raise » ne doit pas être associée à une chaîne de caractère.

Justification

Cette possibilité tolérée jusqu'à présent ne sera plus maintenue dans les prochaines versions du langage.

Exemple

Ne plus écrire :

```
If erreur :
    raise « Le programme a rencontré une erreur »
```

7.7. DYNAMIQUE

Sans objet.

7.8. INTERFACES

Int.FoncUsage	Un script python doit proposer une fonctionnalité d'aide en ligne associée à l'option « --help ».
M=3;F=0;P=110;V=1	
Quelconque	

Description

Par convention, l'option « --help » est définie dans la plupart des scripts. Elle renseigne sur la bonne utilisation et les fonctionnalités que le script met à disposition. Elle est le pendant du manuel utilisateur des applications plus complexes.

Justification

Cette règle améliore la facilité d'utilisation du script.

Exemple

Sans objet.

7.9. QUALITE

QA.EspaceTabulation	Le caractère tabulation est interdit. Le caractère espace doit être utilisé pour l'indentation.
M=3;F=3;P=10;V=2	
Quelconque	

Description

Sans Objet.

Justification

La tabulation doit être évitée, elle n'a pas la même longueur sur tous les éditeurs. Elle peut néanmoins être utilisée si elle est configurée en fonction d'un nombre d'espaces.

Exemple

Sans Objet.

QA.OptionTabulation	L'option « -tt » de l'interpréteur doit être utilisée.
M=3;F=3;P=11;V=0	
Applet	

Description

Sans Objet.

Justification

Cette option détecte le mélange de tabulations et d'espaces qui dégrade la lisibilité du code.

Exemple

Sans objet.

QA.Instruction	Le caractère « : » ne doit pas être suivi d'une instruction sur la même ligne.
M=3;F=1;P=75;V=2	
Quelconque	

Description

On ne doit pas écrire les instructions de traitement sur la même ligne que les expressions if/elif/else/for/while/def/class/except.

Justification

Cette règle améliore la lisibilité du code.

Exemple

Le code suivant n'est pas très lisible :

```
If coordX > valSeuil : calculerMatriceErreur()
For x in listeVal : total += x
While time < 10 : t = delay()
```

Il est préférable de l'écrire comme suit :

```
If coordX > valSeuil :
    calculerMatriceErreur()
For x in listeVal :
    total += x
While time < 10 :
    t = delay()
```

QA.Parenthèses	Les expressions conditionnelles servant à assigner une variable doivent être encadrées par des parenthèses.
M=3;F=2;P=42;V=0	
Quelconque	

Description

L'assignation doit s'écrire : `X = (valeur_vrai if condition else valeur_faux)`

Justification

Cette règle améliore la lisibilité du code.

Exemple

```
# Première version : pas de parenthèse
niveau = 1 if debug else 0
# Seconde version : avec parenthèses
niveau = (1 if debug else 0)
```

La première version est difficile à comprendre car ambiguë à la lecture : le lecteur peut regrouper « niveau = 1 », « if debug », « else 0 ».

7.10. AUTRES REGLES

Gen.Yield	L'instruction yield n'est pas autorisée dans le block try d'une construction try/finally.
M=1;F=3;P=12;V=1	
Quelconque	

Description

Sans Objet.

Justification

Dans une telle construction, il n'y a aucune garantie quant à l'exécution des instructions contenues dans le bloc finally.

Exemple

Sans Objet.

Dec.Declaration	La déclaration d'une méthode statique doit se faire par un décorateur.
M=2;F=2;P=75;V=0	
Quelconque	

Description

Sans Objet.

Justification

La déclaration d'une méthode statique avec l'ancienne syntaxe est moins lisible.

Exemple

L'exemple suivant n'est pas à suivre car dans le cas d'une méthode statique longue, sa déclaration en statique tout à la fin n'est pas très visible.

```
class C:
   def meth (cls):
       ...

   meth = classmethod(meth)   # Rebind name to wrapped-up class method
```

Préférer l'utilisation d'un décorateur :

```
class C:

   @classmethod
   def meth (cls):
       ...
```

Dec.Limitation	Même si l'empilement des décorateurs est autorisé, leur utilisation doit être limitée.
M=2;F=2;P=75;V=0	
Quelconque	

Description

Sans Objet.

Justification

Lisibilité et compréhension.

Exemple

Sans Objet.

Bib.Duplication	On ne doit pas développer des fonctionnalités déjà offertes par la bibliothèque de référence Python.
M=3;F=1;P=24;V=0	
Quelconque	

Description

Sans Objet.

Justification

Economie de ressources pour le développement et la maintenance du code.

Exemple

Sans Objet.

Py.Python3000	Tout nouveau développement doit être conforme à la syntaxe de Python 3.0.
M=3;F=1;P=25;V=1	
Quelconque	

Description

Sans Objet.

Justification

La syntaxe de Python 3.0 étant différente de Python 2.*, sa prise en compte évite de lourds problèmes de portage à terme.

Exemple

Sans Objet.

8. AUTRES ASPECTS SPECIFIQUES AU LANGAGE

Sans objet

9. SYNTHESE

9.1. TABLE RECAPITULATIVE DES REGLES

Les règles sont récapitulées ici, classées par ordre alphabétique.

ANNEXE : EXEMPLE DE PRESENTATION DE CODE PYTHON

L'exemple ci-dessous présente une structure souvent utilisée dans les codes écrits en Python. L'ordre est le suivant : ligne de démarrage, documentation du module, importations des modules, déclarations de variables globales, déclarations des classes, déclarations de fonctions, programme principal. Il illustre aussi l'utilisation des docstring pour commenter le module, la classe, la méthode, l'attribut, la fonction.

```
#!/usr/bin/env python

""" Ce module est un module de test """

#_________________ IMPORT _____________________________
import sys
import os

#_________________ Variables Globales _____________
test = True


#_________________ Definition de classes ___________

class MaClasse(object):
     """ MaClasse est une classe de test
     """

     __totalInstances = 0
     """ __totalInstances est un attribut de classe
     """

     def __init__(self,nom) :
           self.__nomInstance=nom
           """ __nomInstance est un attribut d'instance
           """

           self.__class__.__totalInstances += 1
           print 'Il y a ', self.__class__.__totalInstances,' intances'
           print 'La derniere instance s appelle ', self.__nomInstance


#_________________ Fonctions globales _____________
def test() :
     """  test est une fonction de test
     """
     if test :
           print 'fonction de test'
                instance1 = MaClasse("Framboise")
                instance2 = MaClasse("Myrtille")


#_________________ Main ____________________________
if __name__ == '__main__' :
     test()
```

REFERENTIEL NORMATIF REALISE PAR :

Centre National d'Etudes Spatiales

Inspection Générale Direction de la Fonction Qualité

18 Avenue Edouard Belin

31401 TOULOUSE CEDEX 9

Tél. : 05 61 27 31 31 - Fax : 05 61 28 28 49

CENTRE NATIONAL D'ETUDES SPATIALES

Siège social : 2 pl. Maurice Quentin 75039 Paris cedex 01 / Tel. (33) 01 44 76 75 00 / Fax : 01 44 46 76 76
RCS Paris B 775 665 912 / Siret : 775 665 912 00082 / Code APE 731Z