CNES
STANDARDS REFERENCE
RNC

Reference: **RNC-CNES-Q-HB-80-535**
Version 1
11 May 2009

Manual

CODING RULES FOR PYTHON LANGUAGE

APPROVAL of Standardisation Office	**BN no. 54 dated 16/09/09**

DOCUMENT ANALYSIS PAGE

TITLE: CODING RULES FOR PYTHON LANGUAGE	
KEYWORDS: Language, Script, Python, Rule, Standard, Programming	
EQUIVALENT STANDARD: None	
REMARKS: None	
ABSTRACT: This document sets out the programming rules applicable to any project that includes scripts developed in the Python language.	
DOCUMENT STATUS: This document is part of the collection of approved Manuals in the CNES Standards Reference. It is affiliated to standard RNC-ECSS-Q-80 "Software Quality Assurance".	
NUMBER OF PAGES: 24	**Language:** English (translated from the original French)
SOFTWARE PACKAGES USED / VERSION: Word 2007	
MANAGING DEPARTMENT: General Inspectorate and Quality Directorate (IGQ)	
AUTHOR(S): Anne-Thérèse NGUYEN (DCT/AQ/SO) **PROOFREADING / CHECKING:** Jean-Charles Damery	

DOCUMENT REVISION SHEET

VERSION	DATE	PAGES MODIFIED	REMARKS
1	11/05/2009		Following FEB 91/2009 accepted in BN no. 54 dated 16/09/09, document introduced in the CNES Standards Reference (RNC) of manual RNC-CNES-Q-HB-80-535.

TABLE OF CONTENTS

1. INTRODUCTION

The document "Coding Rules for Python language " is affiliated to standard RNC-ECSS-Q-80 "Software Quality Assurance". It sets out the applicable rules for scripts developed using the Python language.

2. PURPOSE

The aim of this manual is to establish the rules and recommendations for using the Python language to develop elaborate scripts. These rules and recommendations have been drawn up based on the “state of the art” and the “lessons learned” accumulated over the projects.

This document is not a reference manual for the Python language, nor does it address the interactive aspects of the command interpreter. Knowledge of the Python language is a pre-requisite for using this document.

The reader should note that the vocabulary used for Python differs slightly from that used for other object-oriented languages. The definitions of the terms used herein must be learned in order to avoid incomprehension when reading this manual.

3. SCOPE

This document applies to information system development and maintenance, as regards sections programmed in Python. It is supplemented by the document "Règles communes pour l’utilisation des langages de programmation" [Common rules for using programming languages] (AD1).

Follow the procedure below to use the Python rules on a project:

- Select the common rules, the Python rules and the rules applicable to the project according to tailoring criteria; this selection should be made using the tailoring tool.
- Adapt certain rules to the project.

The document is thus intended for more than one type of reader:

- The Project Manager who must correctly specify the Python application to be developed,
- The Project Manager and/or Quality Engineer, whose responsibility is to select the rules and possibly adapt and complete them in accordance with the project context,
- The personnel responsible for completing the project: who must apply the selected rules.

4. DOCUMENTS

4.1. REFERENCE DOCUMENTS

RD	Identification	Title
(RD1)	RNC-ECSS-Q-ST-80	Software Product Assurance

4.2. APPLICABLE DOCUMENTS

AD	Identification	Title
(AD1)	RNC-CNES-Q-HB-80-501	Common rules for using programming languages

4.3. OTHER DOCUMENTS

- Core Python Programming, 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. TERMINOLOGY

5.1. GLOSSARY

Term	Definition
Docstring	A docstring is a string of characters in quotation marks that is used to describe the code. A docstring is typically used to describe a module, a class, a function or method, a data attribute or a variable.
Module	A Python module contains blocks of code to be executed, as well as function and class declarations. Generally, all files containing Python code are considered as modules. The file name is consequently the same as that of the module completed by the file extension ".py". A module may be executed as script or be imported into other Python code. Unlike in other languages, for which classes may be imported, in Python, only modules may be imported.
Package	A packages groups together a series of modules and sub-packages. In particular, each package or sub-package contains a module _init_, which handles initialisation during import.
Object	A Python object is an element with a unique identifier (memory address), type and value. It may be a number, character string, list, function, method, module, class, file etc. If objects have attributes, these attributes are accessed using pointed notation.
Class	A class is a programmed form of real-world abstraction. In practical terms, in Python, it is a data structure with behavioural characteristics. It plays the role of a template (model) for creating "real" objects, called instances. A class has both data attributes and function attributes (methods).
Class attribute	A class attribute is a data attribute or method that may be accessed without having to create an instance. This name is equivalent to the "static" concept, which is more commonly used in C++ and Java.

5.2. ABBREVIATIONS

OO	Object-Orientated

6. COMPLIANCE WITH RELEVANT SPECIFICATIONS AND STANDARDS

The rules set out in this document conform to version 2.5 and prepare Python version 3.0.

7. RULES

7.1. CODE ORGANISATION

Org.InstrucClasse	Class instructions must all be contained in the class method definitions.
M=3;R=1;P=43;V=1	
Any	

Description

A Python class may contain code that is not in the method. This code is consequently executed when the module containing the class is loaded. This possibility must be avoided of language must be avoided.

Justification

This rule enhances code readability and complies with object-orientated programming encapsulation.

Example

The following example should be avoided:

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

Prefer the use of:

```
class ClassName:
       def function(self) :
          <instruction-1>
           .
           .
           .
          <instruction-N>
```

Org.DeclAttributs	All data attribute declarations for a class must be grouped together.
M=3;R=1;P=43;V=1	
Any	

Description

A Python class data attribute must not be declared just anywhere in the class. An area containing all declarations must be defined.

Justification

This rule enhances code readability.

Example

The following example should be avoided, because it declares the data attributes __coordX, __coordY, __coordZ and __phi in different areas:

```
class ClassName :


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

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

Use the following example, in which all data attributes are declared in the initialisation method:

```
class ClassName :

        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	A local variable for a function (block resp.) must be declared at the beginning of the function (block resp.).
M=3;R=1;P=43;V=1	
Any	

Description

Local variables must not be declared just anywhere in the function or block. They must be declared at the beginning of the function.

Justification

This rule enhances code readability.

Example

The following implementation should be avoided:

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

Prefer the use of:

```
def calculationPhi(alpha) :
   phi = 0
   theta = 0
```

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

Org.Elif	The form elif must be used in place of else: if.
M=3;R=0;P=75;V=2	
Any	

Description

Python does not feature the control structure `switch` , which is available in Java or C++. Use of a `switch` in Python involves using several nested if / else if structures. The elif form must be used to code this type of control structure.

Justification

Using `elif` reduces the level of indentation and enhances code readability.

Example

The following implementation should be avoided:

```
if xSquared < 0 :
    print “xSquared is negative – reset”
    xSquared = 0
else :
    if xSquared == 0 :
        print “ xSquared is null"
   else :
       if xSquared > 0 :
          print “ xSquared is positive"
       else :
          raise ...
```

Prefer the use of:

```
if xSquared < 0 :
    print “xSquared is negative – reset”
    xSquared = 0
elif xSquared == 0 :
    print “ xSquared is null"
elif xSquared > 0 :
    print “ xSquared is positive"
else :
    raise ...
```

7.2. CODE PRESENTATION

Pr.Comment	Modules, classes, methods, attributes and functions must be commented using docstrings.
M=2;R=1;P=79;V=2	
Any	

Description

As with any program, regardless of its language, code must be commented in order for it to be maintained. Python offers various ways to include comments in the code: "#", as with most Unix-based script languages, and/or docstrings. In addition to these functional comments, each module, class, function or attribute must be specifically commented using the docstring mechanism.

Justification

Docstrings are special strings of characters. They may be accessed during execution and may be used to automatically generate documentation, much like "javadoc" is used for the Java language.

Example

See Appendix.

7.3. IDENTIFIERS

Id.Attributs	In a class, a data attribute must not have the same name as a method attribute.
M=3;R=3;P=4;V=1	
Any	

Description

Not Applicable

Justification

If a data attribute and a method attribute have the same name in a class, the data attribute will mask the method attribute, which will subsequently never be called.

Example

Not Applicable

Id.AttributPrive	In a class, the name of a private attribute must begin with two underscores.
M=2;R=2;P=30;V=1	
Any	

Description

Not Applicable

Justification

This mark allows the interpreter to distinguish private attributes from other attributes.

Example

Not Applicable

Id.Fichier	A file containing Python code must have the extension ".py".
M=2;R=0;P=105;V=2	
Any	

Description

Not Applicable

Justification

The extension ".py" is used by convention. It allows Python code to be distinguished from code written in another language.

Example

Not Applicable

7.4. DATA

Don.DeclAttribut	All data attributes must be private by default.
M=2;R=1;P=45;V=1	
Any	

Description

In Python, class attributes are public by default. They may be directly accessed in the module or in another module that imports this module. To ensure data encapsulation, every data attribute that does not require specific visibility is explicitly declared as private.

Justification

This rule improves data encapsulation. Furthermore, declaring private attributes avoids conflict with derived class name spaces: during execution, private attributes are renamed to include the name of the class.

Example

In Python, a private attribute has a name that begins with a double underscore (__).

```
# Declaration of a private attribute of a class
class MyClass :
    def __init__(self) :
        __num = 0
```

During execution, the attribute may only be accessed using the name of the attribute changed into `_MyClass__num`.

Don.VarCoherente	The data stored by the same variable must be physically/typologically consistent throughout the entire program.
M=3;R=2;P=80;V=0	
Any	

Description

Python does not restrict the reuse of a variable in a program for storing a different type of data. However, this possibility must not be used.

Justification

Enhances code readability and reliability.

Example

Do not write:

```
angle = 90
...
angle = "rectangle"
...
```

Prefer the use of:

```
angle = 90
...
angleName = "rectangle"
...
```

Don.ListeCoherente	Data stored in a list must be physically/typologically consistent.
M=3;R=2;P=80;V=0	
Any	

Description

An extension of the rule Data.ConsistentVar to lists.

Justification

Enhances code readability and reliability.

Example

Do not write:

```
angles = [0, "rectangle", 180, 360]
```

Prefer the use of:

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

7.5. PROCESSING

Tr.Definition	All the methods of a class must be explicitly defined within the class.
M=3;R=2;P=45;V=0	
Any	

Description

The Python scripting language is quite permissive. For example, it is possible not to explicitly define a method within a class by pointing it to a global function (see example).

Justification

Declaring a method outside the scope of class definition has some disadvantages:

- It impacts code readability;
- It is not compatible with the packaging concept used in OO languages;
- It adds a global function and occupies naming space by adding a new name.

Example

The following example should not be used:

```
# Method defined outside of class
# using a global function

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

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

Use the following implementation:

```
# Method defined explicitly in the class

class C:

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

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

7.6. ERROR MANAGEMENT

Err.ExpressionExcept	The "except : " instruction should not be used without specifying the exception that will be treated.
M=1;R=2;P=95;V=2	
Any	

Description

Python allows the "except : " instruction to be used alone. However, this possibility must not be used.

Justification

Using this instruction without specifying the exceptions causes all signals from the system, including control signals, to be intercepted.
This rule allows the scope of exception processing to be managed.

Example

Not Applicable

Err.ExcepChaine	An exception must not be a character string.
M=2;R=1;P=95;V=2	
Any	

Description

The "raise" instruction must not be associated with a character string.

Justification

This possibility, which was tolerated up until now, will not be maintained in subsequent versions of the language.

Example

Do not write:

```
If error :
    raise "The program encountered an error"
```

7.7. DYNAMIC

Not applicable.

7.8. INTERFACES

Int.FoncUsage	Python scripts must offer an online help function associated with the "--help" option.
M=3;R=0;P=110;V=1	
Any	

Description

By convention, the "--help" option is defined in most scripts. It provides information on correct use and on the script functions available. It serves as an online user manual.

Justification

This rule makes the script even easier to use.

Example

Not applicable.

7.9. QUALITY

QA.EspaceTabulation	Use of the tab character is prohibited. The space character must be used for indentation.
M=3;R=3;P=10;V=2	
Any	

Description

Not Applicable

Justification

Tabs must be avoided, because the tab character is not the same length in all editors. It may be used if the tab is configured according to a specific number of spaces.

Example

Not applicable

QA.OptionTabulation	The interpreter's "-tt" option must be used.
M=3;R=3;P=11;V=0	
Applet	

Description

Not Applicable

Justification

This option detects the combination of tabs and spaces, which adversely impacts code readability.

Example

Not applicable.

QA.Instruction	The ":" character must not be followed by an instruction on the same line.
M=3;R=1;P=75;V=2	
Any	

Description

Processing instructions must not be written on the same line as the expressions if/elif/else/for/while/def/class/except.

Justification

This rule enhances code readability.

Example

The following code is not very readable:

```
If coordX > Threshval : calculateMatrixError()
For x in Vallist : total += x
While time < 10 : t = delay()
```

It should be written as follows:

```
If coordX > Threshval :
    calculateMatrixError()
For x in Vallist :
    total += x
While time < 10 :
    t = delay()
```

QA.Parenthèses	Conditional expressions used to assign variables must be enclosed in parentheses.
M=3;R=2;P=42;V=0	
Any	

Description

Assignment must be written as: `X = (value_true if condition else value_false)`

Justification

This rule enhances code readability.

Example

```
# First version : no parentheses
level = 1 if debug else 0
# Second version : with parentheses
level = (1 if debug else 0)
```

The first version is difficult to understand because it is ambiguous: the reader may group together "level = 1", "if debug", "else 0".

7.10. OTHER RULES

Gen.Yield	The "yield" instruction is not authorised in the try block of a try/finally construction.
M=1;R=3;P=12;V=1	
Any	

Description

Not Applicable

Justification

In this type of construction, the execution of the instructions contained in the finally block cannot be guaranteed.

Example

Not Applicable

Dec.Declaration	A static method must be declared by a decorator.
M=2;R=2;P=75;V=0	
Any	

Description

Not Applicable.

Justification

Declaring a static method using the former syntax is less readable.

Example

The following example should not be used because with long static methods, the static declaration at the very end is not very visible.

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

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

Prefer the use of a decorator:

```
class C:

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

Dec.Limitation	Decorator stacking is allowed but its use must be limited.
M=2;R=2;P=75;V=0	
Any	

Description

Not Applicable.

Justification

Readability and comprehension.

Example

Not Applicable.

Bib.Duplication	Functionalities that are already provided in the Python reference library must not be developed.
M=3;R=1;P=24;V=0	
Any	

Description

Not Applicable.

Justification

Saves resources for code development and maintenance.

Example

Not Applicable.

Py.Python3000	All new developments must conform to Python 3.0 syntax.
M=3;R=1;P=25;V=1	
Any	

Description

Not Applicable.

Justification

Because Python 3.0 syntax differs from that of Python 2.*, taking this into account will prevent eventual significant porting problems.

Example

Not Applicable.

8. OTHER SPECIFIC LANGUAGE ASPECTS

Not applicable

9. SUMMARY

9.1. RULE SUMMARY TABLE

The rules are summarised below, in alphabetical order.

APPENDIX: SAMPLE PYTHON CODE PRESENTATION

The example below features a structure that is often used in written Python code. The order is as follows: starting line, module documentation, module importing, global variable declarations, class declarations, function declarations, main program. It also illustrates the use of docstrings to comment modules, classes, methods, attributes and functions.

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

""" This module is a test module """

#_________________ IMPORT _____________________________
import sys
import os

#_________________ Global Variables _____________
test = True


#_________________ Class Definition __________

class MyClass(object):
     """ MyClass is a test class
     """

     __totalInstances = 0
     """ __totalInstances is a class attribute
     """

     def __init__(self,name):
           self.__nameInstance=name
           """ __nameInstance is an instance attribute
           """

           self.__class__.__totalInstances += 1
           print 'There are', self.__class__.__totalInstances,' instances'
           print 'The last instance is called ', self.__nameInstance


#_________________ Global Functions _____________
def test() :
     """  test is a test function
     """
     if test :
           print 'test function'
                instance1 = MyClass("Raspberry")
                instance2 = MyClass("Blueberry")


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

STANDARDS REFERENCE PRODUCED BY:

Centre National d'Etudes Spatiales

Inspection Générale Direction de la Fonction Qualité

18 Avenue Edouard Belin

31401 TOULOUSE CEDEX 9

Tel.: +33 (0)5 61 27 31 31 - Fax: +33 (0)5 61 28 28 49

CENTRE NATIONAL D'ETUDES SPATIALES

Headquarters: 2 pl. Maurice Quentin 75039 Paris cedex 01 / Tel: +33 (0)1 44 76 75 00 / Fax: +33 (0)1 44 46 76 76

Paris Trade & Companies Registry No. B 775 665 912 / Business Registration No: 775 665 912 00082 / Business Sector Code 731Z