Padrão para a codificação em Python

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Esparso é melhor que denso.
Legibilidade conta.
Casos especiais não são especiais o suficiente para quebrar as regras.

The Zen Of Python

Considerações gerais

O paradigma de programação à ser usado é o de Orientação a Objetos.

Lembrem-se que o código será lido mais vezes do que escrito, logo, devemos dar prioridade à legibilidade. Se você aprendeu a fazer 500 coisas em uma linha só, não faça, isso só será legível para você (durante os 15 segundos que você levou a escrever).

Cada linha deverá conter apenas uma **ÚNICA** instrução.

O uso de ponto-e-vírgula é **TERMINANTEMENTE PROIBIDO**, já que são desnecessários e só é utilizado para escrever múltiplas instruções na mesma linha.

O código deverá ser escrito **TODO** em inglês. A língua escolhida foi definida através de uma votação que aconteceu por e-mail. Os comentários serão em português.

Modules

Ao criar um 'module', devemos incializá-lo com a string especial para documentação, chamada a partir de agora de DocString, deste modo:

```
"""
Módulo que contém a classe Account que será usada como classe
base no Sistema Bancário
"""
```

Também devemos incluir os 'atributos' __author__e __date__, como no exemplo abaixo:

```
__author__= "gvc"
__date__ = "29/08/08"
```

O atributo __author__ deverá conter o login de quem criou o module.

O atributo __date__ deverá conter a data da última versão no formato: DD/MM/YY .

Os imports devem estar localizados acima da DocString e dos atributos __author__e __date__ e deverão estar em linhas separadas. A não ser que se deseje importar mais de uma coisa do mesmo module.

Variáveis

Não se deve atribuir valores de tipos diferentes a uma mesma variável.

O nome das variáveis deverá seguir o padrão visto em IP.
Exemplo: variavelComUmBomNome
variavelcomumnomeruim
variavel_com_um_nome_pessimo

Não se deve usar abreviações.
Exemplo:

```
acc = Account("1234-5", "Guilherme Carvalho")
account = Account("1234-5", "Guilherme Carvalho")
```

Não se deve usar siglas a não ser que a sigla seja mais comum que o nome.
Exemplo:

hypertextTransferProtocolHandler

HTTPHandler

Classes

Classes de Modelo devem ser definidas em seus próprios arquivos, uma por arquivo. Classes que serão RequestHandler's deverão estar no mesmo Module e agrupadas por assunto.
Exemplo: No Module CompanyController vamos ter as classes AddCompany, EditCompany ...

Ao declararmos uma classe devemos definir, nesta ordem:

1. O(s) construtor(es),
2. E então os métodos.

Ao definirmos o construtor, devemos definir seus atributos precedidos por um underline. Devemos definir o get e o set para cada atributo, e usarmos a função property para 'linkar' os métodos à variável que vamos acessar de fora. Exemplos: (ver anexo)

Os métodos de uma classe deverão estar organizados em ordem de importância, métodos com a mesma importância deverão estar em ordem alfabética crescente (de A a Z).

As especificações referentes aos 'modules' também são referentes às classes. Ou seja, devemos definir a DocString e os 'atributos' __author__e __data__.

Exceções

Como são classes, exceções deverão seguir o padrão para classes acrescentando as seguintes observações:

1. Seus nomes devem acabar com Exception, por exemplo a exceção a ser lançada quando um tipo inválido é passado como parâmetro deverá se chamar TipoInvalidoException.
2. Variáveis do tipo de uma exceção deverão ter o nome do tipo eX, onde X pode ser omitido, ou algum número. Por exemplo, um bloco try catch:

```
    try:
        blablabla
    except TipoInvalidoException e, OutraCoisaInvalidaException
e2:
        print "aaaah"
```

Métodos e Funções

Deverão conter a DocString, explicitando qual a funcionalidade do método. A DocString poderá ser simples, caso a função ou o método sejam simples e seus nomes já descrevam bem o que ele fazem. Para métodos que definem um algoritmo complexo, a DocString deverá explicar como o algoritmo funciona e o que ele faz.

Sues nomes devem sugerir ações,

Métodos que forem usar um tipo específico de parâmetros, deverão usar o **type-checking** module.

Espaço em branco

Uma linha em branco deverá separar métodos entre si nas classes.
Não deve haver espaço:

- entre o nome de um método e sua lista de parâmetros. Na declaração e na chamada.
- após (, [ou { para criação de tuplas, listas e dicionários.
- antes),], } para finalizar a criação dos tipos acima citados.
- para alinhar a inicialização de variáveis.
- para dar um valor default a um parâmetro.

Deve haver espaço:

- Após uma vírgula.
- Após ':' em um dictionary.
- antes e depois de um operador binário.
- antes e depois do operador de atribuição.

Anexo

Segue aqui um module que segue os padrões descritos acima:

```
from InvalidTypeException import InvalidTypeException

"""
Classe que representa uma conta.
Atributos: numero, saldo e nome do dono.
"""

__author__ = "gvc"
__date__ = "04/09/08"

class Account:

    number = None
    ownerName = None
    balance = None

    def __init__(self, number, ownerName):
        self.number = number
        self.ownerName = ownerName
        self.balance = 0
        print "Account: " + self.number + "\nOwner: " + self.ownerName

    def credit(self, value):
        """Metodo que adiciona o value para o saldo da self Account."""
        if self.validValue(value):
            self.balance = self.balance + value
        else:
            raise InvalidTypeException(value + " isn't a numeric type")

    def debit(self, value):
        """Metodo que subtrai o value do saldo da 'self' Account."""
        if self.validValue(value):
            self.balance = self.balance - value
        else:
            raise InvalidTypeException(value + " isn't a numeric type")

    def transfer(self, destinyAccount, value):
        """Metodo que debita o valor passado como parametro na 'self'
Account e credita este mesmo valor Account passada como parametro."""
        if self.__class__ == destinyAccount.__class__:
            self.debit(value)
            destinyAccount.credit(value)
        else:
            raise InvalidTypeException(destinyAccount + " isn't an
Account")

    def validValue(self, value):
        return type(value) == type(0.0) or type(value) == type(0)
```

```
if __name__ == "__main__":
    account = Account("1234-5", "Guilherme Carvalho")
    account.credit(10.5)
    print account.balance
```

Referências

http://www.python.org/dev/peps/pep-0008/
http://oakwinter.com/code/typecheck/index.html