



Sebastian Krieger
Roberto de Almeida

Guia
 python
para
oceanógrafos
em quatro dias

TM

Sebastian Krieger e Roberto de Almeida

Guia Python para oceanógrafos em quatro dias

Uma alternativa código livre para análise de dados

20 de abril de 2011

Python é marca registrada da *Python Software Foundation*. Este guia foi produzido utilizando
 \LaTeX e KOMA-Script.

Sumário

Prefácio	ix
1. Engatinhando	1
1.1. Instalação	1
1.2. Lógica de programação	2
1.3. Primeiros comandos	3
1.4. Tipos de variáveis	4
1.5. Algoritmos	4
1.6. Meu primeiro script	4
1.7. Exercícios	4
2. Primeiros passos	5
2.1. Funções	5
2.2. Bibliotecas	5
2.3. Algumas operações	5
2.4. Estatísticas simples	5
2.5. Guia de estilo	5
2.5.1. Leiaute	5
2.5.2. Importações	6

Sumário

2.5.3. Espaços em branco	8
2.5.4. Comentários	9
2.5.5. Documentação	10
2.5.6. Versões	11
2.5.7. Convenções de nomenclatura	11
2.6. Exercícios	12
3. Juntando pedaços	13
3.1. Operadores lógicos e condicionais	13
3.2. Iterações	13
3.3. Manipulação de dados	13
3.3.1. Salvando e carregando	13
3.4. Visualização	13
3.5. Exercícios	13
4. Avançando	15
4.1. Tratamento de dados	15
4.1.1. Média móvel	15
4.1.2. Tendência e ajuste de funções	15
4.1.3. Interpolação	15
4.2. Visualização elaborada	15
4.2.1. Perfis	15
4.2.2. Contornos	15
4.2.3. Histogramas	15
4.3. Exercícios	15
5. Fluência	17
5.1. Análise espectral	17
5.2. Mapas	17

5.3. pyDAP	17
5.4. Exercícios	17
A. Colinha	21

Prefácio

O uso de ferramentas computacionais para a aquisição, visualização, análise e interpretação de dados científicos é imprescindível para as ciências dos mares devido o enorme volume de informação disponível. Neste contexto, Python aparece como uma ferramenta moderna, flexível e de baixo custo. O objetivo deste guia é dar uma breve introdução à linguagem de programação Python aplicada à análise de dados oceanográficos através de exemplos práticos e realistas.

Python é uma linguagem de programação de alto nível de uso geral, interpretada, orientada a objetos, cuja filosofia de design enfatiza a leitura de código. Lançada por Guido van Rossum em 1991, possui atualmente um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos *Python Software Foundation* (PSF). Python não é uma linguagem específica para análise de dados ou estatística. É uma linguagem bas-

tante flexível que pode ser utilizada em diversas aplicações como Internet, processamento de imagens, jogos, etc. Existem inúmeras bibliotecas disponíveis para Python que延伸dem as funcionalidades da linguagem, dentre elas para uso científico. Aqui focaremos em algumas aplicações oceanográficas.

Planilhas eletrônicas convencionais são ferramentas importantes para a análise de dados e estatística. Entretanto com um volume de dados extenso a automação no tratamento destes dados pode ser limitada ou até mesmo impossível em tais planilhas. É importante salientar que, independentemente da ferramenta utilizada, todas elas possuem limitações quanto a sua aplicação a determinados problemas, e Python não é exceção.

A elaboração deste guia foi motivada pela vontade de difundir a linguagem Python como alternativa de qualidade, flexível, livre e gratuita a programas para análise de dados ci-

entíficos comerciais, como MATLAB® e IDL® por exemplo. Todo o conteúdo é baseado em informação amplamente disponível na Internet, tanto nas páginas oficiais do Python quanto nas bibliotecas utilizadas e em fóruns específicos. Quando possível, referências importantes serão indicadas. As principais análises apresentadas são baseadas em métodos sugeridos por Emery e Thomson 2001.

Este guia possui uma página na Internet com conteúdo atualizado, erratas, exemplos, informações adicionais e um fórum para o envio de dúvidas e sugestões. A página pode ser acessada através do caminho regeirk.com/python.

Prerequisites

Este guia não prevê quaisquer conhecimentos prévios em programação ou na linguagem Python. No entanto um curso introdutório à computação certamente traz grandes benefícios, principalmente quanto à lógica de programação. Downey 2008 dá uma boa introdução de como pensar como um cientista da computação utilizando Python de maneira simples, construtiva e intuitiva. Outra boa re-

ferência para desenvolvedores mais experientes é Pilgrim 2004.

Muitos conceitos de programação serão omitidos para manter o texto conciso. Ao invés de descrever estes conceitos, exemplos passo-a-passo ilustram a funcionalidade de diversas aplicações na análise de dados oceanográficos de forma prática e construtiva. Quando apropriado, referências para consulta são indicadas em notas laterais.

Hoje em dia é imprescindível algum conhecimento básico de inglês, principalmente na compreensão de textos. As próprias linguagens de programação são baseadas em inglês e seus principais recursos de documentação disponíveis na Internet são neste idioma. Podem ocorrer casos em que a documentação é internacionalizada e também disponível em português, mas assume-se algum conhecimento básico de leitura de textos no idioma da Rainha ou do Tio Sam para consultar as referências aqui citadas com naturalidade.

Os exemplos

Os códigos de exemplo de cada seção são escritos em forma de tutorial, encorajando a construção de aplicativos em etapas e para

compreender como os algoritmos funcionam. Os exemplos devem ser acompanhados passo-a-passo e executados de maneira interativa para que você comprehenda o funcionamento. Os algoritmos apresentados são simples e podem ser facilmente extendidos de diversas formas de acordo com as particularidades de cada tipo de análise.

Todos os códigos fonte utilizados estão disponíveis em formato digital na página Internet deste guia indicada anteriormente. Entretanto, recomenda-se que o código seja digitado manualmente para a familiarização com a sintaxe da linguagem e para melhor fixação dos conceitos apresentados.

Porque Python?

Existem inúmeros motivos para se aprender a programar em Python e mais tantos para não se aventurar nela. A seguir são apresentados alguns dos motivos para iniciar-se nesta linguagem de programação.

Conciso. Código escrito em linguagens dinâmicas, como Python, tende a ser menor que código escrito em outras linguagens ‘mainstream’. Ou seja, há menos digitação ao

testar os exemplos. No entanto, mais importante que escrever menos, também simplifica a assimilação do algoritmo na mente para realmente compreender seu funcionamento.

Fácil de ler. Pessoas com experiência de programação em outras linguagens são capazes de ler código em Python e compreender o que ele deve fazer.

Extensível. Por padrão, Python vem com inúmeras bibliotecas, incluindo aquelas para funções matemáticas, manipulação de texto e conexão com a Internet. As demais bibliotecas utilizadas neste guia, como para manipulação de dados e geração de gráficos, são livres e simples de baixar, instalar e utilizar.

Interativo. Ao trabalhar com um exemplo, é útil experimentar as funções enquanto elas são escritas, sem ter de escrever outro programa apenas para teste. Python é capaz de executar tanto programas diretamente da linha de comando quanto chamar funções, criar objetos e testar pacotes de forma interativa.

Múltiplos paradigmas. Python suporta estilos de programação orientados a objetos, procedurais ou funcionais.

Multiplas plataformas e livre. Python possui uma única implementação de referência para as principais plataformas disponíveis e é livre e gratuito para todas elas. Os códigos descritos neste guia funcionam nas plataformas Windows, Linux e Mac.

Alternativas

Python não é a única opção de código livre ou gratuita para análise de dados. Qualquer linguagem é capaz de criar uma aplicação e solucionar os problemas que desejamos. No entanto, a melhor escolha depende do esforço necessário para tal solução e a sensação de sucesso obtida com ela. Uma lista com alternativas comerciais e livres pode ser encontrada na página da Wikipedia. Dentre as alternativas gratuitas disponíveis, destacamos as seguintes:

Octave. É uma linguagem computacional desenvolvida para computação matemática e compatível com MATLAB®. Possui uma

interface em linha de comando para a solução de problemas numéricos, lineares e não-lineares. Criado por John W. Eaton, faz parte do projeto GNU.

Scilab. Software científico para computação numérica semelhante ao MATLAB®. Ele fornece um poderoso ambiente computacional aberto para aplicações científicas. Atualmente o projeto é mantido pelo *Scilab Consortium*.

scilab.org

R. Linguagem e ambiente de desenvolvimento integrado para cálculos estatísticos e representações gráficas. Foi criada originalmente por Ross Ihaka e por Robert Gentleman. R é uma linguagem e ambiente similar ao S. Embora com importantes diferenças, muitos códigos escritos para S rodam inalterados em R.

r-project.org

GDL. Implantação livre e código aberto da linguagem de dados interativa IDL®.

gnudatalanguage.
sourceforge.net

Ocean Data View (ODV). Pacote integrado para a exploração, análise e visualização de perfis ou sequências de dados oceanográficos ou georeferenciados.

odv.awi.de

Visão geral

Este guia foi dividido em cinco partes para servir como base para um minicurso introdutório de Python aplicado à oceanografia com quatro dias de duração. Cada parte é motivada por exemplos de aplicação reais que de fácil compreensão e adaptação.

Capítulo 1, *Engatinhando*. Dá instruções de como instalar e executar Python pela primeira vez e explica como executar alguns comandos básicos e um pouco sobre tipos de variáveis. As instruções dadas neste capítulo devem ser seguidas antes do início do curso, para garantir que as necessidades técnicas mínimas sejam satisfeitas e que cada participante tenha todos os aplicativos necessários instalados.

Capítulo 2, *Primeiros passos*. Introduz o conceito de funções e bibliotecas. Demonstra operações aritméticas e estatísticas básicas como soma, produto, exponenciação, mínimo, máximo, média e desvio padrão.

Capítulo 3, *Juntando pedaços*. Ilustra os conceitos de operadores lógicos e condicio-

nais e também iterações. Começa a trabalhar com um volume de dados maior e demonstra como carregar e salvar dados de arquivos. Visualiza estes dados pela primeira vez.

Capítulo 4, *Avançando*. Mostra exemplos de tratamento básico de dados como média móvel e representações gráficas mais elaboradas.

Capítulo 5, *Fluência*. Demonstra recursos mais avançados como análise espectral através da transformada de Fourier e visualização de mapas.

Convenções

Sobre os autores

Sebastian é mestre em ciências em oceanografia física formado pelo Instituto Oceanográfico (IO) e também é bacharel em física formado pelo Instituto de Física (IF), ambos da Universidade de São Paulo (USP). Participou da primeira turma do curso de especialização em oceanografia observacional no *Bermuda Institute of Ocean Sciences* (BIOS), promovido

regeirk.com

pela *Partership for the Observation of the Global Oceans* (POGO) com o apoio da *Nippon Foundation* (NF). Atualmente é aluno de doutorado do programa de pós-graduação do IO-USP. Apesar de não possuir formação específica em computação, tem contato com programação há mais de 15 anos e já trabalhou com as linguagens BASIC, Visual Basic, C, FORTRAN, HTML, JavaScript, SQL, PHP e ASP. Começou a utilizar Python em 2006 para gerar todas as análises apresentadas em sua dissertação de mestrado.

dealmeida.net

Roberto é doutor em oceanografia física formado pelo Instituto Oceanográfico (IO) da Universidade de São Paulo (USP) e oceanógrafo formado pela FURG.... Atualmente é pesquisador do Centro de Previsão de Tempo e Estudos Climáticos (CPTEC) do Instituto Nacional de Pesquisas Espaciais (INPE) em Cachoeira Paulista. ...

Agradecimentos

1. Engatinhando

1.1. Instalação

A primeira coisa que você deve fazer antes escrever seu primeiro programa em Python é instalá-lo em seu computador. As distribuições Linux mais populares já vêm com Python instalado por padrão. O Mac OS em suas versões mais recentes também possui uma versão instalada. Entretanto, se você utiliza Windows, terá que instalar o Python em seu computador. Há diversas maneiras simples para instalá-lo e na maioria das vezes muito bem documentadas. Acesse o sítio Internet em python.org, procure pelas instruções de instalação e siga-as. Não se esqueça também de instalar as bibliotecas utilizadas ao longo deste guia, listadas mais adiante.

Se você não quiser aventurar-se na instalação individual do Python e as principais bibliotecas utilizadas neste guia, recomenda-

mos uma alternativa: *Enthought Python Distribution*. É uma distribuição de Python gratuita para fins não comerciais que vem com uma série de bibliotecas e ferramentas instaladas. Apesar da instalação do Python ser relativamente simples, a instalação de determinadas bibliotecas pode não ser uma tarefa trivial para pessoas menos experientes. Para evitar surpresas e nos certificarmos que você tenha uma versão do Python compatível com este guia, preparamos uma máquina virtual com todos os aplicativos, bibliotecas, códigos fonte e conjuntos de dados aqui utilizados. Para saber mais a respeito desta máquina virtual e instruções para sua instalação, acesse a página Internet deste guia.

Este guia é baseado em Python versão 2.6 e com as seguintes bibliotecas instaladas:

SciPy. Ferramentas científicas para Python (*Scientific tools for Python*) é uma bibli-

www.enthought.com

regeirk.com/python

scipy.org

1. Engatinhando

oteca que possui manipulação de matrizes n-dimensionais rápida e conveniente. Ela fornece uma série de rotinas numéricas amigáveis para integração, otimização numérica e análise estatística avançada.

matplotlib.sourceforge.net

matplotlib. Biblioteca para criação de gráficos e figuras de alta qualidade em uma variedade de formatos. Também possui ferramentas para a criação de mapas através do complemento basemap.

numpy.scipy.org

NumPy. Pacote fundamental para computação científica com Python. Numpy adiciona uma ferramenta rápida e sofisticada para manipulação de matrizes à linguagem Python. Ela é requisito para as duas bibliotecas citadas acima e é instalada automaticamente ao instalar-se qualquer uma delas.

www.imr.no/~bjorn/python/seawater/

seawater. O pacote *seawater* fornece funções básicas para determinar propriedades físicas da água do mar como densidade, calor, etc.

pydap.org

PyDAP. Biblioteca criada por Roberto de Almeida que implanta o protocolo de acesso

a dados DAP, também conhecido por DODS e OPeNDAP.

1.2. Lógica de programação

O objetivo deste guia não é transformá-lo em um guru em programação, mas introduzir conceitos importantes de uma ferramenta para análise de dados oceanográficos. No entanto é necessário desenvolver uma forma de pensar e organizar as idéias de modo a traduzi-las em uma linguagem computacional. Esta forma de pensar combina elementos matemáticos, de engenharia e das ciências naturais. Assim como os matemáticos, cientistas da computação utilizam linguagens formais para denotar idéias (principalmente cálculos computacionais). Como os engenheiros, eles criam coisas, montam componentes para formar sistemas e avaliam os benefícios de alternativas. Como cientistas, eles observam o comportamento de sistemas complexos, formam hipóteses e testam previsões (Downey 2008).

Uma das habilidades mais importantes a serem desenvolvidas é a capacidade de solucionar problemas de maneira lógica. Mas, para solucioná-los, é preciso saber formular estes

problemas, usar sua criatividade para encontrar soluções e expressar estas soluções de maneira clara e acurada.

A forma mais simplificada de se ver um programa é considerá-lo uma sequência de instruções que especificam como executar cálculos computacionais. Estes cálculos computacionais podem ser puramente matemáticos, como o cálculo da média de um conjunto de números, ou simbólicos, como agrupar uma lista de espécies por gênero e organizá-la em ordem alfabética.

As instruções de qualquer linguagem de programação podem ser agrupadas de diferentes formas ou de acordo com a sua função:

Entrada. Adquire dados a partir do teclado, de um arquivo ou outro dispositivo.

Saída. Exibe dados ou resultados na tela, armazena-os em um arquivo ou os envia a outro dispositivo.

Matemática. Performa operações matemáticas como adição ou multiplicação.

Execução condicional. Verifica por determinadas condições e executa a sequência de instruções apropriada.

Repetição. Performa ação repetitiva, em geral com alguma variação.

Por mais simplista que possa parecer, qualquer programa, independente de sua complexidade, pode ser resumido por instruções de um destes cinco grupos. Desta forma, a programação consiste em dividir um problema grande e complexo em subtarefas cada vez menores e simplificadas até que elas possam ser executadas por instruções básicas. E este é talvez o maior desafio para qualquer programador, compreender o problema a ser resolvido e subdividí-lo em tarefas distintas.

1.3. Primeiros comandos

Inicie o seu interpretador Python. Dependendo da versão instalada aparecerá na tela texto semelhante ao seguinte,

```
Python 2.6.5 (r265:79063, Apr 16 1
2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits"
or "license" for more information
.
>>>
```

```
>>> print '1 + 1 = ', 1 + 1  
1 + 1 = 2
```

1.4. Tipos de variáveis

- números - texto formatando text (- list - tuple
- dictionary - array

1.5. Algoritmos

1.6. Meu primeiro script

You will be working with Python interactively in this chapter, so you should save recommendations.py somewhere where the Python interactive interpreter can find it. This could be in the python/Lib directory, but the easiest way to do it is to start the Python interpreter in the same directory in which you saved the file.

1.7. Exercícios

Exercício 1.1. Utilize um navegador Internet para acessar a página python.org. Esta página possui informações sobre a linguagem Python, atalhos para páginas relaci-

onadas a Python e também possibilita a busca em sua documentação.

Faça uma busca pelo termo `print` e visite a primeira página do resultado. Explique o conteúdo em suas palavras.

Exercício 1.2. Seja dado o seguinte conjunto de valores: [1, 10, 8, 6, 5, 3, 2, 5, 5, 3]. Determine a soma, os valores mínimo e máximo e a média da série.

2. Primeiros passos

2.1. Funções

2.2. Bibliotecas

2.3. Algumas operações

2.4. Estatísticas simples

- Média - Desvio padrão - Mínimo - Máximo -
Mediana

2.5. Guia de estilo

Um aspecto importante na hora de se escrever código em qualquer linguagem é a legibilidade, principalmente levando-se em conta que código é lido com maior frequência que escrito. Por isso Guido van Rossum e Barry Warsaw sugerem um guia estilo para a formatação de código na linguagem Python.

A íntegra em inglês deste guia pode ser consultada no atalho ao lado . Ele foi elaborado para garantir consistência no código escrito. Atenção na formatação do código e na sua documentação é importante para consultas futuras, principalmente para justificar determinadas soluções adotadas quando não podemos contar com nossa memória. Um resumo do guia de estilo citado é apresentado a seguir.

2.5.1. Leiaute

Indentação

Cada nível de indentação deve ser composto por quatro (4) espaços. O uso de tabulações não é recomendado e não se deve misturar tabulações com espaços. Os principais editores disponíveis possuem recursos para a utilização de espaços ao invés da tabulação.

Indentação com 4
espaços

2. Primeiros passos

Linhas de no
máximo 79
caracteres

Extensão das linhas

As linhas de código não devem possuir mais de 79 caracteres. Muitos dispositivos de exibição são limitados a 80 caracteres cuja quebra de texto pode distorcer a estrutura visual do código, dificultando a sua compreensão. Para blocos de texto longos (documentação ou comentários), recomenda-se o limite de 72 caracteres por linha. Seguindo estas recomendações pode-se, em dispositivos de maior resolução, utilizar múltiplas janelas de código lado a lado para consulta simultânea.

Continuação de linha implícita do Python pode ser feita entre parênteses (()), colchetes ([]) ou chaves ({ }). Linhas compridas podem ser divididas envolvendo-se expressões entre parênteses, mantendo a identação apropriada. A região preferida para quebra de operadores binários é após o operador ao invés de antes dele. O código 2.1 ilustra um exemplo de quebra de linhas.

Linhas em branco

Separe as definições de funções e de classes de nível superior com duas linhas em branco. Definições de métodos dentro de uma classe são separadas por uma única linha. Linhas

em branco extras podem ser utilizadas moderadamente para separar grupos de funções relacionadas. Nas funções utilize linhas em branco para indicar seções lógicas.

Codificação de caracteres

Todo o código Python deve utilizar a codificação de caracteres ASCII ou *Latin-1* (ISO-8859-1). Para Python 3.0 ou superior a codificação preferida é UTF-8.

2.5.2. Importações

Comandos de importação devem estar em linhas separadas, como por exemplo em

✓ **import** os
import sys
import numpy

Deve-se evitar

✗ **import** os, sys, pylab

No entanto, para a importação de funções de uma mesma biblioteca pode-se utilizar

✓ **from** numpy **import** array, cos, pi

As importações devem ser listadas no topo do arquivo, logo abaixo da documentação e

Código 2.1 Exemplo ilustrativo de quebra de linhas

```
1  class Rectangle(Blob):
2
3      def __init__(self, width, height,
4          color='black', emphasis=None, highlight=0):
5          if (width == 0 and height == 0 and
6              color == 'red' and emphasis == 'strong' or
7                  highlight > 100):
8              raise ValueError("sorry, you lose")
9          if width == 0 and height == 0 and (color == 'red' or
10              emphasis is None):
11              raise ValueError("I don't think so---values are %s, %s" %
12                  (width, height))
13      Blob.__init__(self, width, height,
14          color, emphasis, highlight)
```

2. Primeiros passos

dos comentários do módulo. Elas devem ser agrupadas seguindo a seguinte ordem:

1. Módulos da biblioteca padrão;
2. Módulos de terceiros relacionados;
3. Módulos locais e/ou específicos para a aplicação.

Deve-se separar grupos de importação por uma linha em branco. Qualquer especificação `_all_` deve ser feita após as importações.

2.5.3. Espaços em branco

Em expressões ou declarações, evite espaços em branco estranhos nas seguintes situações:

- Imediatamente dentro de parênteses, colchetes ou chaves;

```
✓ fruta(maca[1], {pera: 2})  
✗ fruta( maca[ 1 ], { pera: 2 } )
```

- Imediatamente antes de uma vírgula, ponto-e-vírgula e dois-pontos;

```
✓ if x == 4: print x, y; x, y = y,  
    x  
✗ if x == 4 : print x , y ; x , y =  
    y , x
```

- Imediatamente antes de parênteses que inicia a lista de argumentos de uma chamada de função;

```
✓ fruta(1)  
✗ fruta (1)
```

- Imediatamente antes de parênteses de abertura que inicia uma indexação ou corte;

```
✓ dict[ 'chave' ] = list[ indice ]  
✗ dict [ 'chave' ] = list [ indice ]
```

- Mais de um espaço ao retor de um operador de atribuição para alinhamento.

```
✓ x = 1  
y = 2  
var_longa = 3  
  
✗ x           = 1  
y           = 2  
var_longa = 3
```

Outras recomendações:

- Sempre cerque os seguintes operadores binários com um espaço simples de cada lado: atribuição (`=`), atribuição de incremento (`+=`, `-=`, ...), comparações (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`), **in**, **not in**, **is**, **is not**), booleanos (**and**, **or**, **not**);
- Utilize espaços ao redor de operadores aritméticos;

```

✓ i = i + 1
enviado += 1
x = x * 2 - 1
hipot2 = x * x + y * y
c = (a + b) * (a - b)

✗ i=i+1
enviado +=1
x = x*2 - 1
hipot2 = x*x + y*y
c = (a+b) * (a-b)

```

- Não utilize espaços ao redor do sinal de `=` quando indicar um argumento de palavra-chave ou um valor de parâmetro padrão;

```

✓ def complexo(real, imag=0.0):
    return magico(r=real, i=imag)

```

```

✗ def complexo(real, imag = 0.0):
    return magico(r = real, i =
                  imag)

```

- O uso de instruções compostas (múltiplas instruções na mesma linha) não é recomendado;

```

✓ if oque == 'blah':
    fazer_blah()
fazer_um()
fazer_dois()
fazer_tres()

```

```

✗ if oque == 'blah': fazer_blah()
fazer_um(); fazer_dois();
fazer_tres()

```

2.5.4. Comentários

Comentários são utilizados para inserir anotações no módulo que sejam legíveis a qualquer programador. Eles são incluídos para facilitar a compreensão do código. Um bom comentário não pode contradizer o código escrito. Tenha cuidado em manter os comentários atualizados quando houver mudança no código. Eles devem ser sentenças

completas respeitando as normas gramaticais e terminando com um ponto final. Para código de uso pessoal ou restrito, os comentários podem ser em qualquer idioma. Para códigos mais abrangentes, recomenda-se redigir os comentários em inglês.

Em bloco Comentários em bloco geralmente aplicam-se a todo ou parte do código que se segue. Eles devem ser identados no mesmo nível deste código. Cada linha de um comentário em bloco inicia com um caractere `#` e um espaço simples. Parágrafos dentro do bloco são separados por uma linha contendo apenas um `#` devidamente identado.

Em linha Utilize comentários em linha com moderação. Este comentário deve estar na mesma linha da declaração, separado de ao menos dois espaços. Ele inicia com um caractere `#` e um espaço simples. Comentários em linha podem ser desnecessários e até distrair quando afirmam o óbvio:

```
x = x + 1      # Incrementa x
```

Entretanto, o seguinte exemplo pode ser útil:

```
✓ x = x + 1      # Compensa borda
```

2.5.5. Documentação

Escreva a documentação (*docstrings*) para todos os módulos, funções, classes e métodos públicos. Ele é a sequência literal que segue a declaração de um módulo, função, classe ou método. Esta sequência torna-se o atributo especial `__doc__` deste objeto. Documentação não é necessária para métodos não-públicos, entretanto deve haver uma descrição sobre a funcionalidade do método. A sequência deve estar entre `"""..."""` como ilustrado a seguir:

```
def complexo(real=0.0, imag=0.0):      1
    """Forma um numero complexo.        2
                                               3
    Argumentos:
        real -- a parte real (padrao      4
               0.0)
        imag -- a parte imaginaria (      5
               padrao 0.0)
    """
...
```

Note que o trio de aspas `"""` que fecha a documentação está em uma linha por si só e é preferencialmente precedida por uma linha em branco. Para o caso de texto descritivo de uma linha apenas, as aspas de fechamento

podem estar na mesma linha.

2.5.6. Versões

Manter registro e controle de revisões e versões é importante para certificar-se de que se está trabalhando com a versão mais atualizada. As seguintes linhas devem ser incluídas após a sequência de documentação do módulo e antes de qualquer outro código, separado por uma linha em branco acima e abaixo. Além de manter o controle de versões de cada módulo, sistemas de controle de revisão como *Subversion* ou *CVS* podem propagar atualizações automaticamente.

```
--version__ = "$Revision: -1234-$"  
# $Source$
```

2.5.7. Convenções de nomenclatura

Há diversos estilos de nomenclatura e é importante ser capaz de reconhecer cada um deles, independentemente de seu uso. Distingue-se comumente os seguintes estilos:

- b – letra minúscula simples;
- B – letra maiúscula simples;

- minúsculas;
- minúsculas_com_traco_baixo;
- MAIÚSCULAS;
- MAIÚSCULAS_COM_TRAÇO_BAIXO;
- IniciaisMaiúsculas ou *CamelCase*. Note que, neste caso, as abreviações devem ser utilizadas na forma maúscula (ex.: `HTTP-ServerError`);
- caixaMista;
- Palavras_Capitalizadas_Com_Traço_Baixo.

As seguintes formas especiais que utilizam o traço baixo como prefixo ou sufixo também são reconhecidas:

- _traço_baixo_inicial – indicador de uso interno. Por exemplo, `from M import *` não importa os objetos cujo nome inicia com um traço baixo;
- traço_baixo_final_ – utilizado por convenção para evitar conflitos com palavras-chave do Python (ex.: `Toplevel(master, class_='ClassName')`);

2. Primeiros passos

- `_traço_baixo_inicial_duplo` – ao nomear um atributo de classe, invoca a deformação de seu nome;
- `_traço_baixo_inicial_e_final_duplos_` – objetos ‘mágicos’ que existem em espaços de nome controlados pelo usuário (ex.: `_init_`, `_import_` ou `_file_`). Nunca invente estes nomes, utilize apenas como documentado.

2.6. Exercícios

Exercício 2.1. Descreva um algoritmo para ...

3. Juntando pedaços

3.1. Operadores lógicos e condicionais

3.2. Iterações

3.3. Manipulação de dados

3.3.1. Salvando e carregando

3.4. Visualização

3.5. Exercícios

Exercício 3.1. Carregue os dados da série temporal do *Bermuda Atlantic Time-series Study* (BATS) e plote um diagrama de temperatura e salinidade (T-S).

4. Avançando

4.1. Tratamento de dados

4.1.1. Média móvel

4.1.2. Tendência e ajuste de funções

4.1.3. Interpolação

4.2. Visualização elaborada

4.2.1. Perfis

4.2.2. Contornos

4.2.3. Histogramas

4.3. Exercícios

Exercício 4.1. Sobreponha ao diagrama T-S do capítulo anterior a feição média de T-S com linha contínua vermelha e 2 pixels de espessura. Transforme este diagrama T-S em um diagrama de estado ao incluir as

isolinhas de densidade no fundo através do comando `contour`.

Exercício 4.2. Visite a página Internet da biblioteca `matplotlib` e pesquise pelo comando `subplot`. Gere uma figura com dois gráficos lado a lado, o da esquerda contendo o diagrama de estado do exercício anterior e o da direita contendo os perfis médios de temperatura e salinidade desde a superfície até 2500 m de profundidade.

`matplotlib`.
`sourceforge.net`

5. Fluência

5.1. Análise espectral

5.2. Mapas

5.3. pyDAP

5.4. Exercícios

Referências

DOWNEY, A. B. *Think Python: An Introduction to Software Design*. Needham, Massachusetts: Green Tea Press, 2008.

EMERY, W. J.; THOMSON, R. E. *Data analysis methods in physical oceanography*. 3rd. ed. [S.l.]: Elsevier Science, 2001. 654 p. ISBN 978-0444507570.

PILGRIM, M. *Dive Into Python*. Apress, 2004. 413 p. ISBN 978-1590593561. Disponível em: <<http://diveintopython.org/>>.

A. Colinha
