
INICIAÇÃO CIENTÍFICA

UM TUTORIAL DE INTRODUÇÃO AO AMBIENTE DE PESQUISA

CREADO POR: JHONATAN A. A. MANCO, LIVIANY P. VIANA

JHONATANJAAM@GMAIL.COM,LIVIANY.METEORO@GMAIL.COM

FELIZ TUTORIAL

Sumário

1	LINUX	2
1.1	Instalação	2
2	Python Anaconda	4
2.1	Instalação	4
2.2	Criação do Ambiente de trabalho com Anaconda	4
2.3	Instalação de Pacotes no Ambiente criado	5
3	Pacotes do Python para aplicações Geofísica	6
3.1	Leitura de arquivos de dados	6
3.2	Pacotes para plotar Mapas	6
3.3	Pacote Xarray	7
3.4	Pacote MetPy	7
3.5	Anacoda WLS	7
4	Dados de reanálises ERA-5	8
5	Git	10
5.1	Clonando o Repositório Criado	12
5.2	Adicionando Arquivos no GIT	12
5.3	Outros comandos uteis no Git	14
6	Gráficos no Python, Mapas	15
6.1	Gráfico temporal	23

Sobre este documento

LINUX

O Linux é um sistema operacional UNIX. Unix significa que sistema operacional portátil (executado independentemente da arquitetura), multitarefas (executa várias tarefas ao mesmo tempo) e multiusuário (permite mais de um usuário ao mesmo tempo).

Alguns sistemas do tipo UNIX são:

- Linux
- MacOS
- Solaris

Já o Linux é um sistema operacional UNIX de Código aberto e gratuito, disponibilizado sob a Licença Pública Geral (GPL) GNU. Qualquer usuário pode executar, estudar, modificar e redistribuir o código-fonte.

1.1 Instalação

As distribuições mais comuns encontradas no ambiente de pesquisa são:

- Ubuntu
- Mint
- Fedora
- Elementary OS

O link leva a pagina de instalação mais recente de cada versão, para propósitos deste tutorial se recomenda a instalação de Ubuntu, umas vez que a distribuição de Linux mais usada e difundida.

Para instalar o linux juntamente com o Windows, basta seguir este tutorial:

Link: [LINUX WINDOWS](#)

o Linux também pode ser usado nas novas versões de Windows(Windows 10 versão 2004 e superior) e é conhecido com o nome de WSL. No seguinte link estão as instruções para a instalação do WSL no ambiente Windows.

Link: [WSL](#)

Para propósitos deste tutorial, o ambiente WLS pode ser suficiente para realizar as tarefas. No entanto, para usuários que irão usar supercomputadores, instalar modelos meteorológicos, trabalhar com Fortran, entres outros, seu uso não é recomendado.

Uma vez instalado o sistema operacional Linux, são apresentados os comando mais usados dentro do terminal do sistema operacional Ubuntu. Para abrir o terminal, basta usar [Ctrl+Alt+T](#), ou procurar nas aplicativos do sistema.

O terminal do Linux é um dos principais características do sistema operacional. Serve para receber os comandos do usuário a partir do teclado e repassa-os diretamente ao sistema operacional.

Os comandos mais usados dentro do terminal são :

- pwd: encontra o caminho completo do diretório atual.
- cd: Acessa uma determinada pasta (diretório)
- ou mudar de diretório atual, ex: cd .., cd /nomedapasta
- ls: Lista todos os arquivos do diretório
- mkdir: Cria um diretório
- **rm: Remove um arquivo/diretório, seu uso não pode ser desfeito, apaga por completo**
- df: Mostra a quantidade de espaço usada no disco rígido
- top: Mostra o uso da memória
- cat: Abre um arquivo
- mv: pode ser usado para mover ou renomear arquivos.
- clear: limpar o terminal se estiver cheio de muitos comandos usados anteriormente, [Ctrl+l](#)
- : caminho do home ex: cd , vai para home
- vi: Abre o editor

Uma vez instalado o Linux e se familiarizado com seus comandos básicos dentro do terminal, na seguinte capítulo será instalado o Python usando o Anaconda.

Python Anaconda

Anaconda é a plataforma de distribuição de Python mais popular. Esta conta com o comando [conda-install](#), que permite a instalação de miles de pacotes de código aberto tanto para Python, R e outras linguagens de programação.

2.1 Instalação

Para sua instalação basta entrar no seguinte link e procurar o pacote adequado para o sistema operacional correspondente:

Link: [Anaconda](#)

2.2 Criação do Ambiente de trabalho com Anaconda

Uma vez instalado, basta configurar o Anaconda com a versão do Python e os pacotes necessário para a tarefa desejada. Para tal objetivo é útil a criação de ambientes dentro do mesmo Anaconda, o que permite ter diferentes configurações de Python para propósitos diferentes (Python2.7, Python3.3, Python3.8).

Neste tutorial será instalada, a modo de exemplo, a versão do Python 3.8, criando o ambiente de nome [py1](#). Para isto abra o Terminal, (*Ctrl+t*), e verifique que versão do Anaconda esta instalada e encontra-se atualizada:

```
Terminal
(base) computernome:~$ conda -V
conda 4.11.0
(base) computernome:~$ conda update conda
computerCollecting package metadata (current_repodata.json): done
```

```
Solving environment: done
...
```

Uma vez verificada a instalação o comando `conda create -n` será usado para criar o ambiente `py1`

```
(base) computernome:~$ conda create -n nomeambiente(py1)
python=x.x(3.8) anaconda
```

onde `nomeambiente` é o nome que terá o ambiente criado, neste caso `py1`, e `python=x.x` define qual versão do Python será usada, podendo ser mudada, a modo de exemplo, para 2.7.

Para ativar o novo ambiente basta usar o comando `conda activate` com o nome do ambiente

```
(base) computernome:~$ conda activate nomeambiente(py1)
(py1) computernome:~$
```

Note que o nome do ambiente(`py1`) substitui o nome `(base)` no terminal.

Para desativar o ambiente basta usar o comando `conda deactivate` com o nome do ambiente .

```
(base) computernome:~$ conda deactivate nomeambiente(py1)
(base) computernome:~$
```

2.3 Instalação de Pacotes no Ambiente criado

Por defeito Anaconda e sus respectivos ambientes criados na seção anterior contam com os pacotes mais usados no Python, tais como: `numpy`, `matplotlib`, `datetime` ... entre outros.

Para instalar os pacotes necessários para as diferentes aplicações do Python para um determinado ambiente pode ser usado o comando:

```
(base)computernome:~$conda install -n nomeambiente pacoteainstalar
```

O instalando o pacote dentro do ambiente desejado:

```
(py1) computernome:~$ conda install -n pacoteainstalar
```

será

Uma vez criado o ambiente de trabalho faze-se necessário instalar os pacotes necessário para um determinado objetivo. Neste caso são pacotes necessários para trabalhar com dados meteorológicos, o que será visto na seguinte seção.

Pacotes do Python para aplicações Geofísica

3.1 Leitura de arquivos de dados

Existem uma grande variedade de arquivos que podem ser abertos com o Python instalando os pacotes necessários. O primeiro tipo de dado abordado serão os dados do tipo **NETCDF**(.nc, .cdf). Este é um dos formatos mais usados para guardar dados meteorológicos, tanto de Reanalises como de saídas de modelos. Para a leitura deste tipo de arquivos Python conta com uma biblioteca chamada **netcdf-python**. Para sua instalação use o comando **conda install**

```
(py1) computernome:~$ conda install -c conda-forge netCDF4
```

3.2 Pacotes para plotar Mapas

Python oferece grande variedade de bibliotecas que facilitam a plotagem de mapas em duas dimensões. Uma das mais usadas é **BASEMAP**. Este pacote é similar ao programas Grads e NCL, muito usados nas Ciências Atmosféricas. Ela depende de outra biblioteca que devem ser instalada antes, **GEOS** (Google Earth Overlay Server), uma vez que usa muitas das funções para criação de mapas desenvolvidas para esta. Sua instalação com o CONDA é simples:

```
(py1) computernome:~$ conda install geos
```

Uma vez instalada o GEOS o Basemap pode ser instalado no CONDA assim:

```
(py1) computernome:~$ conda install basemap
```

Se o comando anterior não funciona use este:

```
(py1) computernome:~$ conda install -c conda-forge basemap
```

Finalmente instale estas ultima atualizações para os pacotes anteriores.

```
(py1) computernome:~$ conda install basemap-data-hires
```

```
(py1) computernome:~$ conda install -c conda-forge nc-time-axis
```

ou usando o pip

```
(py1) computernome:~$ pip install nc-time-axis
```

3.3 Pacote Xarray

A biblioteca Xarray é um pacote que facilita a leitura de dados com dados de N-dimensão (multi-índices), tal como, coordenadas, mapeamento e rótulos, ou seja, ideal para trabalhar com dados de mais de 2 dimensões como os dados meteorológicos provenientes de reanálises (ERA5). Xarray é uma mescla das bibliotecas Pandas e NetCDF, facilitando a leitura e cálculo das variáveis. Para mais informações visite a pagina:

Link: [Xarray](#).

Para sua instalação, use o código abaixo, cuja versão está em v0.11.3.

```
(py1) computernome:~$ conda install -c anaconda xarray
```

3.4 Pacote MetPy

A biblioteca MetPy oferece ferramentas em Python para leitura, visualização e cálculos com dados meteorológicos. Podendo ser muito útil em varias áreas como a termodinâmica uma vez que conta com funções para calculo de variáveis como CIN, CAPE, LCL entre outros e gráficos de temperatura, pressão, radio sondagem, etc. Mais informações no seguinte link

Link: [MetPy](#).

```
(py1) computernome:~$ conda install -c conda-forge metpy
```

Outras funções pode ser instaladas usando o CONDA mas para os objetivos deste tutorial as bibliotecas necessárias para plotar dados meteorológicos já estão instaladas. Os próximos passos serão usar estar para plotar dados de Reanalises.

3.5 Anacoda WLS

Para quem escolho fazer o tutorial com a distribuição de Windows para Linux o seguinte tutorial mostra os passos para instalação do Anaconda no WLS.

Link: [Anaconda WLS](#).

Dados de reanálises ERA-5

Os dados meteorológicos conhecidos por "reanálises", se referem aos dados que descrevem de forma mais precisa o tempo ou clima (lembrando que estes dois termos são diferentes!), ou seja, são dados gerados a partir das observações (uso de assimilação de dados) e previsão de tempo de curto prazo (modelos atmosféricos), utilizando modelos meteorológicos modernos. A partir disso, esses dados ajudam a entender o tempo/clima presente e passado de forma mais clara e robusta.

Um exemplo do banco de dados de reanálises, cita-se as do ERA5, que quer dizer, em inglês *ECMWF Reanalysis, Version 5* oriundo do *Centro Europeu de Previsões Meteorológicas de Médio Prazo* (ECMWF, em inglês). São dados mais robustos e completos com altíssima resolução espacial (31 km) e temporal (cada hora). Mais detalhes podem ser encontrados neste link <https://www.ecmwf.int/en/forecasts/dataset/ecmwf-reanalysis-v5>.

Para obter esses dados de reanálises, segue-se os passos-a-passos abaixo:

1. **Passo 1:** Cadastrar-se nesta página no canto superior direito (login/register);

```
https://cds.climate.copernicus.eu/cdsapp#!/home
```

2. **Passo 2:** Após a realização do **passo 1**, entre com email e senha cadastrado, e busque os dados de seu interesse em *Datasets*. Por exemplo, os dados horário desde 1979 até os dias atuais;

```
ERA5 hourly data on pressure levels from 1979 to present
```

3. **Passo 3:** Clicando nos dados acima, poderás verificar a página com 4 (quatro) abas denominadas:

```
Overview Download data Quality assessment Documentation
```

4. **Passo 4:** Clique na aba *Download data* e aparecerá as opções que debes escolher: tipo de produto, variável, nível de pressão, ano, mês, dia, hora, área geográfica e formato (netcdf), como na .

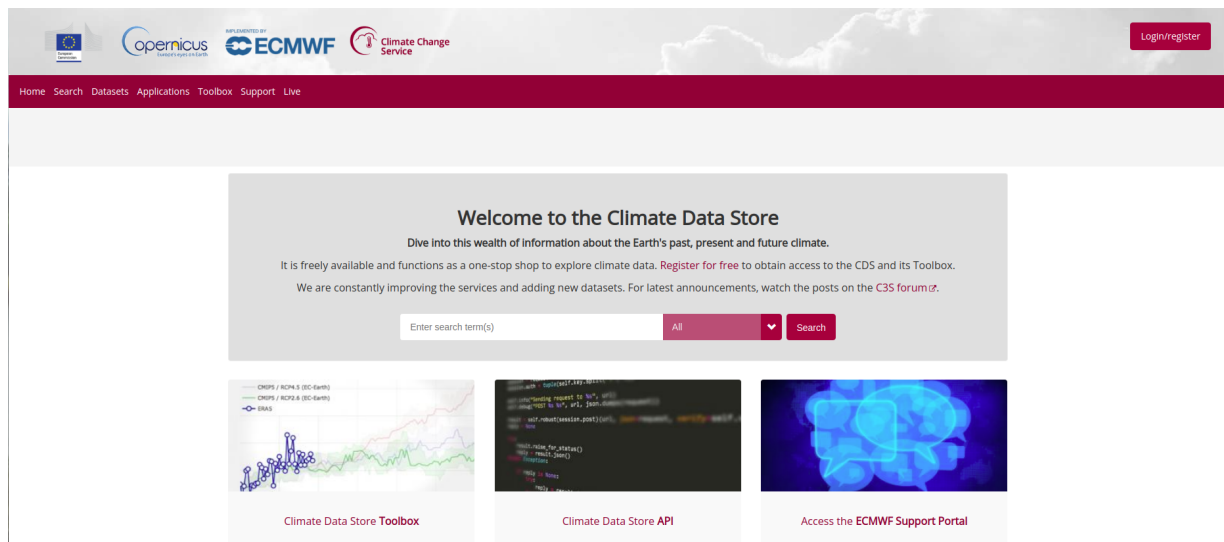


Figura 1: Caption

```
[Product type] [Variable] [Pressure level] [Year]
[Month] [Day] [Time] [Geographic area] [Format]
```

5. **Passo 5:** Após escolher as opções desejadas, clique em:

```
Submit form
```

Git

Uma ferramenta que permite ter controle de versão dos códigos desenvolvidos no Python e dos textos no LATEX é o Git. O controle de versão serve para alterar qualquer tipo de arquivo sempre mantendo um histórico das modificações realizadas, o que permite facilmente voltar a uma versão anterior dos códigos ou dos textos. O Git foi desenvolvido para proporcionar uma plataforma de trabalho para desenvolvimento de códigos por vários usuários, permitindo de um controle das alterações feitas por cada membro do grupo. Além disto o Git oferece uma ampla portabilidade uma que seu acesso é virtual (servidor em nuvem do git) e também permite ser instalado em qualquer computador. Isto proporciona ao usuário a facilidade de alterar seus códigos com grande facilidade em diferentes máquinas de trabalho, além de permitir ter um histórico completo das modificações.

Para trabalhar no Git basta criar uma conta numa das suas tantas plataformas de distribuição. Entre estas plataformas encontra-se o GitHub, BitBucket, entre outras. Estas plataformas irão fazer o controle de versão e guardar na sua nuvem os códigos inseridos. Neste tutorial recomendamos o uso do BitBucket uma vez que os códigos que serão desenvolvidos são para uso privado. No entanto se o objetivo desejado é compartilhar com uma determinada comunidade os desenvolvimentos recomenda-se o uso do GitHub. Para criar uma conta no Bitbucket entre no seguinte link e siga as instruções,

Link: [BitBucket](#)

Crie seu usuário usando um nome de usuário e senha, sem usar um login automático pela conta do Google, já que serão de muita importância para o gerenciamento do repositório.

Uma vez criada a conta, basta abrir um novo repositório, que será uma pasta virtual onde os códigos serão inseridos e controle de versão será feito pelo BitBucket. Para criar um novo repositório clique em qualquer dos círculos vermelhos (signo '+'), veja a figura:

Adicione as informações necessárias

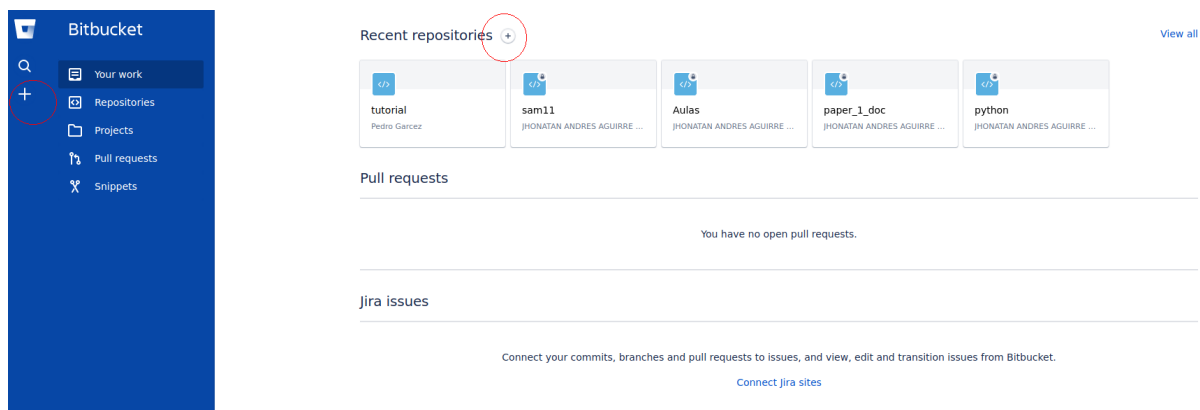


Figura 2: Criação de um repositório no Bitbucket

The image shows the 'Create a new repository' form in Bitbucket. The form has a title 'Create a new repository' and a link 'Import repository'. It contains several fields and options: 'Workspace' (Reactingmx), 'Project' (Select project), 'Repository name' (empty text box), 'Access level' (checked 'Private repository' with a note: 'Uncheck to make this repository public. Public repositories typically contain open-source code and can be viewed by anyone.'), 'Include a README?' (No), 'Default branch name' (e.g., 'main'), and 'Include .gitignore?' (Yes (recommended)). There is a link 'Advanced settings' and two buttons at the bottom: 'Create repository' and 'Cancel'.

Figura 3: Arquivos criados por definição na criação do repositório "exemplo" no Bitbucket

Uma vez criado (recomenda-se o uso do REAME), Fig 4 o repositório está disponível para ser usado salvando os arquivos que precisem de um controle de versão, como relatórios em LATEX, códigos de Python, Fortran, entre outros. Na figura podem se ver dois arquivos que foram criados por definição, o REAME e o .gitignore. O REAME é um arquivo usado para descrever a finalidade do repositório, ajudar o usuário para uso dos códigos ou fazer comentários pertinentes. O .gitignore tem como finalidade excluir arquivos que não precisam do controle de versão seja porque são resultados dos códigos, são dados muito pesados que não podem ser compartilhados na rede (existe um limite máximo dependendo da versão do git), esse arquivos devem ir listados por filas exemplo: excluir o arquivo dados.nc.

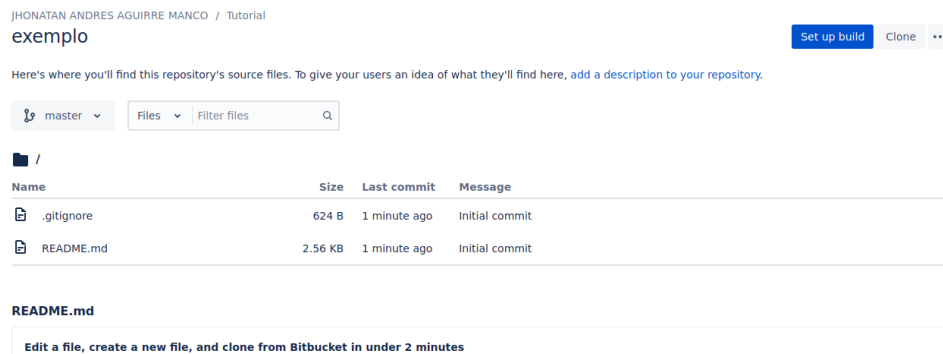


Figura 4: Criação de um repositório no Bitbucket

Além deste arquivos o Bitbucket mostra um pequeno tutorial de como criar um REAME e como inserir arquivos no repositório, o que será abordado seguidamente.

5.1 Clonando o Repositório Criado

No canto superior direito da Fig.4 encontra-se um botão para clonar o repositório, este botão proporciona um link direto para o repositório (https ou ssh). O link criado é único para o repositório e permite criar uma pasta contendo todos os arquivos do repositório em diferentes maquinas ou pastas. Para isto basta executar o link criado no terminal, na maquina e localização onde o repositório será criado:

```
(py1) computername:~local$  
git clone https://usuario@bitbucket.org/usuario/exemplo.git
```

Uma vez clonado basta verificar na pasta o conteúdo da pasta, que vai ser igual ao criado no BitBucket, neste caso terá o REAME e .gitignore.

5.2 Adicionando Arquivos no GIT

Para adicionar arquivos no repositório basta entrar na pasta clonada e como primeiro passo verificar que a versão clonada contenha as últimas atualizações feitas.



Este passo deve ser executado sempre que inicie-se um novo trabalho no repositório para evitar conflitos de versão entre as diferentes pastas clonadas em diferentes computadores. Não trabalhe em diferentes computadores ao mesmo tempo, pode levar a erros de versão no Git.

Para atualizar o GIT:

```
(py1) computernome:~local$  
git pull --all
```

Uma vez atualizado o Git, no primeiro ingresso, serão configurados o e-mail e o usuário para cada máquina na qual o repositório foi clonado. Isto permite que seja identificada em que máquina e quem realizou as alterações. Para isto basta:

```
(py1) computernome:~local$  
git config --global user.email "you@example.com"  
(py1) computernome:~local$ git config --global user.name "Your Name"
```

Uma vez configurado e atualizado o repositório, este está pronto para guardar os códigos e as alterações que serão feitas. Para adicionar um arquivo no repositório use:

```
(py1) computernome:~local$ g  
it add filetoadd
```

Para verificar o processo o Git conta com o comando:

```
(py1) computernome:~local$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    new file:   filetoadd
```

Onde é mostrado, no terminal, a etapa do processo de adição do arquivo, neste caso, 'Changes to be committed'. O seguinte passo é consignar (commit) o arquivo com um comentário pertinente que permita identificar as alterações feitas. Para isto use o comando:

```
(py1) computernome:~local$ git commit -m 'comentario'
```

Como foi feita uma adição de arquivo um comentário pertinente seria 'adicionando arquivo filetoadd'. Quando é modificado um arquivo este comentário permite identificar as alterações do código sem ter que entrar para ver o que foi feito.

Uma vez consignado o estado do repositório pode ser consultado:

```
(py1) computernome:~local$ git status  
On branch master  
Your branch is ahead of 'origin/master' by 1 commit.  
  (use "git push" to publish your local commits)
```

O qual está indicando que nosso repositório está à frente do nosso repositório virtual (o clonado, origin/master), uma vez que este conta com um arquivo novo. Assim, esta mudança deve ser reportada (push) ao repositório principal (origin/master), para o que é necessária usar o seguinte comando:

```
(py1) computernome:~local$ git push origin
```

Se é consultado o estado do repositório, obtém-se :

```
(py1) computernome:~local$  
no changes added to commit (use "git add" and/or "git commit -a")
```

que especifica que repositório está atualizado e que novas alterações podem ser feitas e consignadas no repositório.

Se é consultado o repositório virtual diretamente no BitBucket a alteração feita ficou disponível, neste caso o novo arquivo adicionado deve ser encontrado.



Cabe destacar, que uma vez o repositório virtual seja atualizado, todas as demais versões do repositório devem ser atualizadas para conter as novas mudanças. Se isto não for feito, erros de versão serão criados e terão que ser solucionados usando o comando `git stash`, `git checkout`, `git rm` (consulte o `git help`: [Bitbucket help](#))

5.3 Outros comandos úteis no Git

Para não estar repetindo a senha do repositório cada vez que seja atualizado o repositório basta pedir para o Git guardar a senha usando os seguintes comandos:

```
(py1) computernome:~local$  
git config --global credential.helper store  
git pull
```

Gráficos no Python, Mapas

Uma vez instalado o Python e todas as bibliotecas recomendadas, nesta seção serão mostrando como usar elas para plotar Mapas a partir de dados meteorológicos.

Como ponto de partida foi usado o tutorial [netcdf_python](#), onde os comandos básicos para fazer um mapa no Python são apresentados usando os dados de Reanalises do NCEP/NCAR [**ReanalysisProject**].

Para começar abra um editor de texto, o da sua preferencia, Ubuntu conta com o gedit. Uma vez aberto o editor inicie um novo arquivo o qual tera um nome do código e a extensão do Python (ex.main.step1.py) e cole as seguintes linhas que são comando que o Python ira executar.

- Para importar as bibliotecas necessárias que serão usadas no programa

```
1 NCEP/NCAR Reanalysis -- Kalnay et al. 1996
2 http://dx.doi.org/10.1175/1520-0477(1996)077<0437:TNYRP>2.0.CO;2
3 '''
4 #Importa a biblioteca numpy, que contem funcoes uteis
5 #como: mean, abs, sqrt, entre outras.
6 import numpy as np
7
8 #Importa o Datasets, que permite a leitura de .nc arquivos.
9 from netCDF4 import Dataset
10
11 #Importa a biblioteca matplotlib.pyplot para usar
12 #funcoes para realizar graficos.
13 import matplotlib.pyplot as plt
14
15 # Importa a biblioteca datetime para trabalhar com datas no python.
16 import datetime as dt
17
```



```

18 #Importa a funcao ncdump, que imprimir no terminal
19 #as variaveis e informacoes dos .nc files
20 #from info_ncfiles import ncdump
21
22 #Importa biblioteca necessarias para fazer mapas,

```

As cores no editor de texto facilitam a leitura e programação. Palavras reservadas, comentários, funções, caracteres entre outros, possuem uma cor específica. Uma vez copiadas as linhas e salvo o arquivo com a sua extensão (.py) um bom editor de texto mostra diferentes cores, assim como é apresentado neste documento. Comentários em Python são realizados antecedendo cada linha com o símbolo #, neste tutorial aparecem em cor verde antecedendo e detalhando os comandos de Python.

- Leitura de um arquivo meteorológico usando a função Dataset da biblioteca netCDF4.

```

1
2 from mpl_toolkits.basemap import Basemap, addcyclic, shiftgrid
3
4
5 #1. Load data
6 #####
7
8
9 #Nome do arquivo a ser carregado.
10 nc_f = './ncfiles/air.sig995.2012.nc'
11
12 #Dataset: funcao do netCD4 para leer e carregar um arquivo .nc
13 #como uma classe em python.
14
15 # nome do array criado = Dataset(nome do arquivo, modo leitura('r'))
16 nc_fid = Dataset(nc_f, 'r')
17
18 #Funcao para imprimir as informacoes do arquivo carregado.
19 #Informacoes, como: nome das variaveis no arquivo, dimensoes..
20 #referencia temporal, entre outros...

```

A função `ncdump` é uma adaptação da sua homônima mas para funcionar no Python. Ela serve para imprimir as informações do arquivo carregado, entre elas: Número de dimensões, variáveis disponíveis, descrição dos dados, dados, etc. Isto é sempre necessário uma vez que só serão extraídas da classe as variáveis desejadas.

Para instalar no LINUX, abra o terminal

```
(base) computernome:~$ sudo apt-get install ncdump
```

Uma vez instalado informe o arquivo .nc ou .netcdf(ambos na formatação netcdf) que desejamos abrir:

```
(base) computernome:~$ ncdump nomedoarquivo.nc
```

Outra ferramenta simples para visualização rápida de dados com formato netcdf é o `ncview`

Para instalar no LINUX:

```
(base) computernome:~$ sudo apt-get install ncview
```

O uso de `ncview` é simples, no terminal abra o arquivo a ser visualizado

```
(base) computernome:~$ ncview nomedoarquivo.nc
```

Aparecerá uma interface gráfica, onde é possível selecionar a variável a ser visualizada, assim:

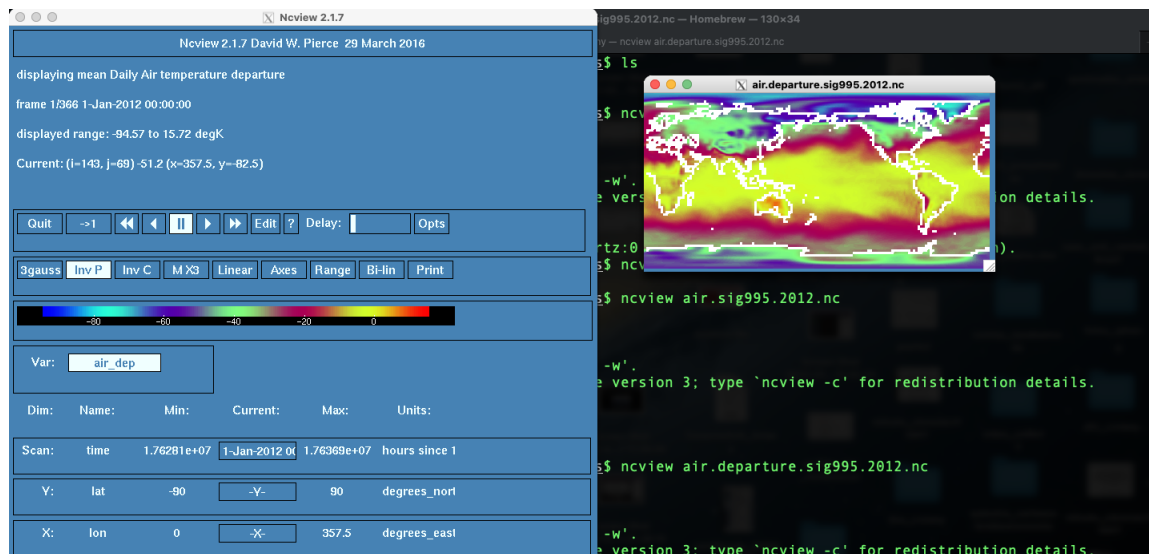


Figura 5: Janela de execução do ncview

Se o arquivo tiver mais de uma variável, selecione uma para criar o gráfico. Para uma vista temporal basta clicar em qualquer ponto do mapa. Automaticamente aparecerá um gráfico temporal com todas as datas desse ponto do mapa.

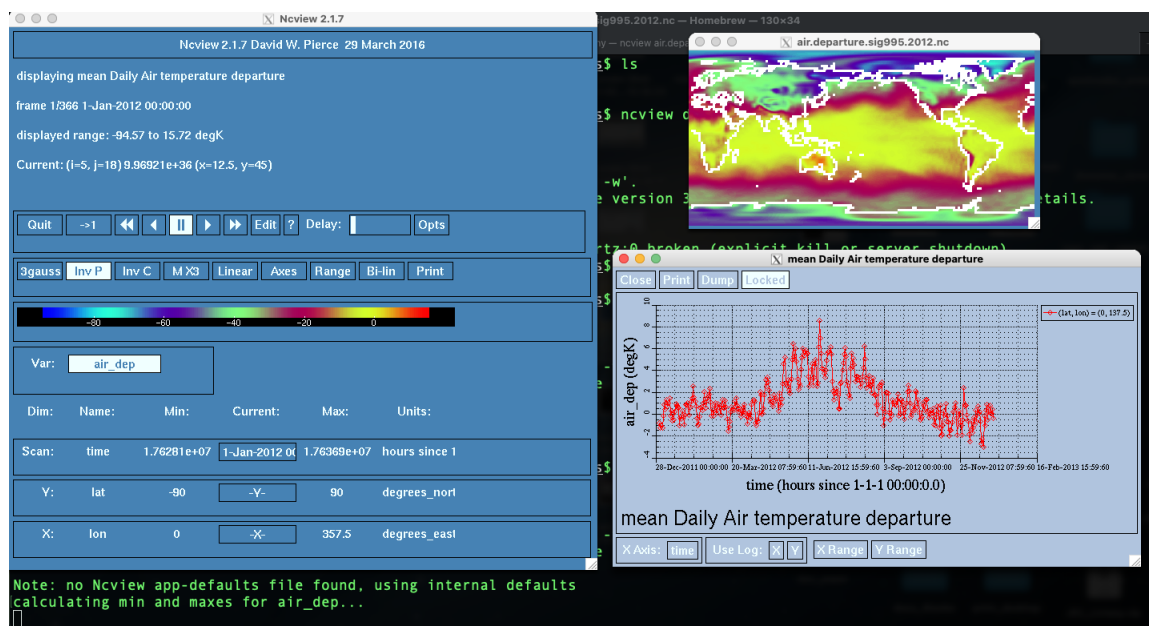


Figura 6: Janela de execução do ncview, com plote temporal

Continuando com o código.

- Extração das variáveis da classe `nc_fid` que serão usadas pelo Python, para fazer cálculos ou para realizar seu plote.

Para este passo é muito útil o `ncdump`, uma vez que não conhecemos as informações do arquivo, como: nome das variáveis, quantas variáveis, tamanho, etc.

```

1 #Neste caso o arquivo tem 3 dimensoes (time, lat, lon)
2 #e uma variavel air que contem a temperatura para o ano de 2012.
3
4 #Para extrair as variveis basta pegar da classe
5 #creada nc_fid o nome da variavel e atribuir o nome desejado
6 #para trabalhar com ela assim:
7
8 lats = nc_fid.variables[ 'lat'][:]
9 lons = nc_fid.variables[ 'lon'][:]
10 time = nc_fid.variables['time'][:]
11 air = nc_fid.variables[ 'air'][:]
12
13 #Os dois pontos [:] significa que serao extraidos todos os dados

```

- Criando o vetor de tempo que contem as datas dos dados.

Uma vez extraídas as variáveis que serão usadas é necessário associar um vetor de tempo a estas, uma vez que são dados temporais (dependentes do tempo) e é o Python oferece um tipo específico de dado para trabalhar com eles. Estes dados podem ser operados facilmente usando as funções da biblioteca `datetime`. Neste arquivo a biblioteca foi carregada nas primeiras linhas como "dt" assim:

```

1 import datetime as dt

```

A modo de exemplo para criar uma data no python usando a função `date` da biblioteca `datetime`:

```

1 data=dt.date(ano, mes, dia )

```

Como os dados carregados do arquivo `.nc` tem informações temporais, salva na variável `time` (data dos dados), faz-se necessário criar um vetor de dados usando este tipo de dado, isto será de muita utilidade na hora de visualizar as datas nos plots.

Para visualizar a variável `time` antes de criar o vetor temporal imprima-la e execute o código, assim:

```

1 print(time)
2 exit()

```

A seguir são mostrados os passos necessários para a criação do vetor temporal.

1. Desfase dos dados

Dependendo do tipo de dado é necessário adiantar ou atrasar as datas, dependendo da referência. No caso deste exemplo os dados da reanálise estão desfasados por 48 horas, o que é necessário levar em conta para criar o vetor de tempo.

```

1
2
3
4 #2.Criando o vetor de tempo para as datas dos dados.
5 #####
6
7 #A funcao dt.timedelta (dt foi nome dado a biblioteca do Python
8 #datetime importando) permite criar um delta de tempo
9 # que pode ser facilmente extraido ou adicionado as datas que
10 #serao criadas.

```

Muitas vezes é necessário desfazer as datas dependendo do fuso horário, o que facilmente também pode ser resolvido com este offset.

2. Definir a referência dos dados

Para criação do vetor de tempo ou de datas, é necessário saber como estão referenciadas as datas do arquivo .nc que foi carregado com a função `Dataset`. Esta informação sempre está disponível no arquivo e pode ser vista usando a função `ncdump`, como foi visto anteriormente. Neste caso as informações do tempo do arquivo oferecem "time : hours since 1-1-1 00:00:0.0" Isto significa que as datas estão referenciadas respeito ao ano=1, month=1, dia=1, hora=0 (Sim!!! o Ano=1 faz 2023 anos). Quem define a referência dos dados é quem faz os dados, por isso muda continuamente e sempre deve ser consultado.

3. Laço para criação do vetor temporal

```
1 offset = dt.timedelta(hours=48)
2
3 # Laço para criação do vetor de datas
4 # Este laço percorre todas os tempos carregados
5 # e salvo na variável time, referenciando-los
6 # a data informada (1/1/1 0:00:00) e subtraindo o offset definido de
7 # 48 horas
8
9 for t in time:
10
11     dt_time = [dt.date(1, 1, 1) + dt.timedelta(hours=t) - offset]
12
13 #Este laço, também pode ser escrito de um forma mais simples:
14
15 #dt_time = [dt.date(1, 1, 1) + dt.timedelta(hours=t) - offset\
```

4. Definir data desejada e encontrar sua posição.

Uma vez criado o vetor de tempo, facilmente pode-se escolher a data desejada para plotar. Para isto defina sua data usando da função `date`:

```
1
2 #mas ambos realizam o mesmo trabalho, criação do vetor de datas.
```

5. Encontrar a posição da data desejada no vetor de tempo

Para encontrar a data escolhida (`my_date`), dentro das disponíveis no vetor de tempo, percorre-se o vetor `dt_time` definido anteriormente usando outro laço (única maneira de percorrer vetores e matrizes). Para saber o tamanho de `dt_time` usa-se a função do Python `len` que devolve um número natural, sendo este o tamanho do vetor. Vale a pena lembrar que os vetores no Python começa desde a posição 0 e os laços terminam numa posição antes da definida. Assim este laço vai percorrendo o vetor temporal desde a posição 0 até o tamanho total -1, procurando no vetor de tempo a data escolhida `my_date`. Para isto serve o condicional "if", quando a data do vetor que esta sendo percorrida `dt_time[i]` coincide com `my_date`, o laço é parado e sua posição é salva na variável `index` definida fora deste.

```

1
2 #Mi data
3 my_date=dt.date(2012,1,24)
4
5 #Laco para saber qual e a posicao no
6 #vetor de tempo da data desejada (my_date).
7
8 #Indice que indica essa posicao.
9 #Inicialmente comeca em 0, uma posicao qualquer.
10 index=0
11
12
13 for i in range(0,len(dt_time)):
14     if(my_date==dt_time[i]):

```

- Realizar o gráfico da variável desejada

Uma vez extraídas as variáveis e criado o vetor temporal que contem a datas dos dados, o seguinte passo é realizar os gráficos das variáveis desejadas, tanto espacialmente (latitude, longitude) como temporalmente (datas).

Para realizar o gráfico espacial, Python conta com uma serie de bibliotecas que permitem realizar mapas (latitude, longitude), neste exemplo estas bibliotecas já foram carregas no inicio do código como:

```

1 #Importa a biblioteca matplotlib . pyplot para usar
2 # funcoes para realizar graficos .
3 import matplotlib . pyplot as plt
4 #Importa biblioteca necessarias para fazer mapas ,
5 # entre ela o basemap
6 from mpl_toolkits.basemap import Basemap, addcyclic, shiftgrid

```

Uma vez que as bibliotecas que serão usadas já foram carregas, basta usa-las para realizar o mapa. Primeiro, abra um novo gráfico no Python com os seguintes comandos

```

1     index=i
2     break
3
4
5 #3)Para graficar os dados de extraido do arquivo .nc
6 #####
7
8 # Abre um figura no Python e assigna o nome fig
9 # Nessa figura e onde o mapa sera gerado e modificado
10 fig = plt.figure()

```

Agora define-se a projeção que vai ser usada para definir o mapa. Esta define como a terra esférica será mostrada no plano. A Biblioteca [Basemap](#) conta com varias opções disponíveis. Neste exemplo será usada a opção 'Robinson'. Define-se também os limites da longitude e da latitude que tera o mapa. Como será plotada a terra completa usam-se os limites máximos.

```

1
2 #Adjust the location of the interior of the figgure
3 fig.subplots_adjust(left=0., right=1., bottom=0., top=0.9)

```

```

4
5 #Determina que projecao vai ser usada.
6 #proj = ccrs.PlateCarree()
7 proj = 'moll'
8 #proj = 'cyl'
9 #proj = 'robin'
10
11 #Define os intervalos das latitudes a ser plotadas (-90,90)
12 #limite inferior latitude
13 lat_i = -90.0
14 #limite superior latitude
15 lat_f = 90.0
16
17 #Definem os intervalos das longitudes a ser plotadas (0,360)
18 #limite inferior longitude
19 lon_i = -180.0

```

Agora, serão usadas as funções disponíveis na Biblioteca [Basemap](#) usando as definições anteriores.

```

1 #limite superior longitude
2 lon_f = 180.0
3
4 #Abre um mapa, seguindo os limites das latitude e longitude
5 #usando a projecao escolhida.
6 #Para outras opcoes:
7 #https://matplotlib.org/basemap/api/basemap_api.html
8 m = Basemap(projection=proj, llcrnrlat=lat_i, urcrnrlat=lat_f,\
9             llcrnrlon=lon_i, urcrnrlon=lon_f, resolution='c', lon_0=180)
10
11 #m = Basemap(projection='moll', llcrnrlat=-90, urcrnrlat=90,\
12 #            llcrnrlon=0, urcrnrlon=360, resolution='c', lon_0=0)
13
14 #Para plotar as linhas dos continentes no basemap de nome m
15 m.drawcoastlines()
16
17 #Para plotar fronteira da projecao do map no basemap de nome m
18 m.drawmapboundary(color='k', linewidth=1.0, fill_color=None, zorder=None
19                 , ax=None)
20
21 #Para plotar os paises nos continentes no basemap de nome m
22 m.drawcountries()
23
24 m.drawparallels(np.arange(-90.,90.,30.),labels=[1,0,0,0]) # draw
    parallels

```

Uma vez definido o mapa deve ser atribuído a este a variável que será plotada. Isto permite adaptar a variáveis escolhida ao formato esférico da terra. A função [addcyclic](#) permite fazer isto usando as longitudes que foram extraídas do arquivo .nc (lons) como referencia. Para mais informações sobre esta e outras funções disponíveis no Basemap visite o [link](#).

A variável escolhida para plotar foi carrega com o nome "air" a qual é uma função de tempo, latitude e longitude (air(t,lon,lat)) e contem as temperatura da superfície para o todo globo no ano de 2012. Aqui

`air[index,:]` representa a temperatura para a posição temporal `index` (definido por `my_date`) para todas longitudes `[:]` e todas as latitudes `[:,:]`. Agora a variável modificada para um formato esférico que contem as informações necessárias para realizar o `plote global` da variável "air" é chamada para "air_cyclic"

```
1 m.drawmeridians(np.arange(-180.,180.,60.),labels=[0,0,0,1]) # draw
   meridians
2
3
4 #Adds a longitude value, and a columns of values to the data array.
5 #Adds cyclic (wraparound) points in longitude to one or several arrays,
6 #the last array being longitudes in degrees. e.g.
7
8 #Adiciona o valor da longitude a variavel que esta
9 #sendo plotada, renomeando ela como air_cicly.
10 #Assignando as longitudes, que sao ciclicas(terra redonda),
11 #a variavel que que ser plotada air[index(my_date),todas latitudes,
   todas longitudes]
12 #Isto e o que define um mapa.
```

A longitude pode ser definida como de 0 a 360 ou de -180 a 180. Igual acontece com o latitude, definida de de 0 a 180 ou de -90 a 90. Dependendo dos limites escolhidos deve a malha deve ser movida para coincidir com eles. Para isso e usada a função `shiftgrid`:

```
1
2
3 # Shift the grid so lons go from -180 to 180 instead of 0 to 360.
4 #Move os limites da malha para adaptar ao limites escolhidos.
5 #Neste caso foi defido de -180 a 180, para o formato 0 a 360.
```

Uma vez que as coordenadas já foram definidas completamente, uma malha bidimensional (latitude,longitude) é criada e definida como duas matrizes, uma para cada coordenada, usando a função `meshgrid`. Note que a para criação da malha usamos a coordenada cíclica definida como "lons_cyclic", uma vez que esta esta associada á uma malha esférica.

```
1 #Neste caso foi defido de -180 a 180, para o formato 0 a 360.
2
3 air_cyclic, lons_cyclic = shiftgrid(180., air_cyclic, lons_cyclic, start
   =False)
```

Definida e criada a malha esférica é necessário projetar estas coordenadas ao plano bidimensional usando a projeção escolhida anteriormente. Este passo já foi feito quando definimos o mapa pela função `Basemap`, onde foi defina a projeção a ser usada, basta extrair esta informações do mapa de nome 'm'.

```
1 # Crea array 2D da grade em formato lat/lon para o Basemap
2 # fazer o mapa
3 lon2d, lat2d = np.meshgrid(lons_cyclic, lats[:])
```

Definida a matriz no plano cartesiano (x,y), a função `contourf` pode ser usada. Esta função é usada para plotar contornos de uma determinada variável, mas antes é necessesario definir que contornos são desejados. Para esto vamos usar a função no numpy(carregado como np no inicio do programa) `linspace`.

```
1 x, y = m(lon2d, lat2d)
2
3
```

```

4 #Define o vetor com os valores dos contornos
5 #desejados a plota

```

Agora podemos usar a função `contourf` com o vetor `v`:

```

1 #Cria o plot de contorno da variavel air_cyclic.
2 #e atualiza a figura que estava aberta.
3 cs = m.contourf(x, y, air_cyclic, v, cmap='RdBu_r', extend='both')

```

Esta função atualiza automaticamente a figura que abrimos anteriormente usando `fig=plt.figure` e chamamos de `fig`.

Finalmente, vamos usar outras funções da biblioteca `matplotlib` carregada no exemplo nas primeira linhas com o nome "plt". Esta permite criar e modificar a barra de cores (`cbar`) usando a função `colorbar`, colocar um titulo a figura usando função `colorbar`, neste caso foram usadas as informações já definidas como o "var_desc", descrição da variável "air" e "my_date" que define a data que foi escolhida para plotar. Para salvar o mapa num determinado formato num arquivo externo foi usando a função `savefig`, que é dos atributos da nossa figura definida como "fig". O nome e formato dados foram "temperatura_global_2012.png".

```

1 #Cria a barra de cores usando o vector v para
2 #definir os limites, neste caso a barra horizontal e
3 #esta escalada, (shrink=0.5).
4 cbar = plt.colorbar(cs, ticks=v, orientation='horizontal', shrink=0.5)
5
6 #Coloca outras informacoes disponives para cada variavel como
7 #descricao=var_desc e unidades na barra de cores.
8 cbar.set_label("%s (%s)" % (nc_fid.variables['air'].var_desc, \
9                             nc_fid.variables['air'].units))
10
11 #Titulo da figura
12 plt.title("%s on %s" % (nc_fid.variables['air'].var_desc, my_date))
13
14
15 #Salva a figura de nome fig em un aquivo.png com o none:
16     temperatura_global_2012
17 #usando uma resolucao de 1000 dpi
18 fig.savefig('temp_brasil.png', dpi=1000)
19
20 # Mostra a figura gerada, se nao simplesmente sera salva como
21 #um arquivo .png
22 plt.show()

```

A função `plt.show()` simplesmente permite que o Python mostre o mapa que esta sendo gerado. Sem esta função o mapa sera salvo no arquivo png sem ser mostrado na execução do programa.

6.1 Gráfico temporal

Uma vez gerado o mapa da temperatura sobre o globo, vamos escolher um ponto específico sobre este, definido pelas coordenadas de latitude e longitude, para ver uma serie temporal da temperatura. Este tipo de gráfico é muito usado para acompanhar o desenvolvimento temporal de uma determinada variável

sobre uma região, como no caso de fenômenos como El Niño Oscilação Sul (ENOS) e índices como o do dipolo do oceano Índico (em inglês, IOD).

Como no exemplo anterior, vamos carregar o mesmo arquivo, extrair as mesmas variáveis e criar o vetor temporal. No entanto, vamos usar a função `num2date` da biblioteca já instalada `netCDF4`.

Para isto carregue a função a ser usada como:

```
1 #Importa a função do datetime que permite transformar
2 #as datas.
3 from netCDF4 import num2date, date2num
```

Carregada a função, esta é usada para criar o vetor temporal com as datas. Mas para isto é necessário definir as unidades do dado temporal que estamos lendo e o tipo de `calendário`.

```
1 data_units='hours since 1-1-1 00:00:0.0'
2 data_calendar='gregorian'
```

Estas unidades são determinadas usando as informações do arquivo de dados `.nc` usando a função `ncdump`. O calendário também é informado no cabeçalho do arquivo.

A função `date2num` faz o caminho contrário, passa do formato de datas `datetime` do Python para o formato numérico referenciado desde uma data inicial, usando as unidades e o calendário definidos.

Uma vez definidas estas informações e criado o vetor temporal nomeado como `dt_time2`:

```
1 dt_time2= num2date(time,units=data_units, calendar=data_calendar)
```

Criado o vetor de tempo, o gráfico temporal pode ser realizado, mas antes é necessário definir que posição espacial vamos a ver.

Para isto vamos definir uma longitude e uma latitude do mapa de temperaturas. O Python conta um tipo de dados chamados de `dicionário`. Os dicionários são úteis para definir palavras-chaves que estarão associadas a uma determinado dado definido pelo criador.

Aqui definimos vários dicionários como:

```
1 cp = {'name': 'Cachoeira Paulista, Brazil', 'lat': -22.39, 'lon':
      -45}
2 dw = {'name': 'Darwin, Australia', 'lat': -12.45, 'lon': 130.83}
3 #dw = {'name': 'Medellin, Colombia', 'lat': 6.23, 'lon': -75}
4 #dw = {'name': 'New York, USA', 'lat': 40.43, 'lon': -73}
```

O dicionário com o nome `cp`, define a localização em latitude e longitude da cidade de Cachoeira Paulista, SP Brasil, usando as palavras-chaves `lat` e `lon`. Já o dicionário com o nome `dw`, define a localização em latitude e longitude da cidade de Darwin, Austrália, usando as palavras-chaves `lat` e `lon`.

Definidas a latitude e longitude que desejamos acompanhar temporalmente, é necessário conhecer a posição dessas coordenadas nos vetores `"lats"` e `"lons"`, carregados previamente.

Para isto vamos fazer uso da função para vetores do Python `argmin`. Esta função retorna em forma de índice a posição de um vetor. Também vamos fazer uso da função valor absoluto da biblioteca `numpy` (`numpy.abs`).

Assim o índice do vetor mais próximo das coordenadas de latitude e longitude escolhidas pode ser encontrado, procurando a menor diferença entre as coordenadas definidas e todas as coordenadas do vetor. Em outras palavras vamos encontrar a distância das nossas coordenadas definidas respeito a todas as outras latitudes e longitudes. A posição onde encontrarmos a menor distância tanto para latitude como longitudes, será a posição desejada.

Em Python esta operação se traduz como:

```

1 lat_idx = np.abs(lats - cp['lat']).argmin()
2 #Encontra a longitude mais perto a desejada e definida no dicionario.
3 lon_idx = np.abs(lons - cp['lon']).argmin()

```

As posições desejadas de latitude e longitude serão armazenadas nas variáveis 'lat_idx' e 'lon_idx'.

Uma vez definido nosso vetor temporal, a data e as coordenadas desejadas, podemos realizar o gráfico temporal.

Para isto abra um figura como o nome de fig no Python usando a biblioteca [matplotlib.lib.pyplot](#)

```

1 fig = plt.figure()

```

Para preencher nossa figura com um plote de linha vamos usar a função [plot](#) da mesma biblioteca.

O gráfico de linha se define com as abscissas, eixos 'x' e 'y'. No eixo x vamos colocar nosso vetor temporal em datas, todas as data(dt_time[:]). No eixo y vamos colocar a temperatura do ar para o todos os tempos e a latitude e longitudes escolhidas.(air[:,lat_idx,lon_idx]).

```

1 plt.plot(dt_time, air[:, lat_idx2, lon_idx2], c='r')

```