python™

# Métodos Computacionais da Astronomia – Estudos dirigidos 3 e 4



Professor: Pedro da Silveira Ferreira

Período: 2020/1 - PLE

# Introdução

Neste estudo dirigido vamos revisar um pouco de python básico, dado que Computação I é pré-requisito desta disciplina e é baseada em python, e introduzir algumas das principais bibliotecas usadas em python. Algumas vamos começar a explorar nesse estudo e outras apenas nos próximos.

**Atenção os links sublinhados em verde são de referências em português.**

# Linguagens de programação

Atualmente as principais linguagens, com principais quero dizer as mais usadas, são **Python**, **Java**, **Javascript**, **C#** e **C/C++**. Para os cientistas as principais linguagens de interesse são **Python**, **C/C++** e **Fortran** (**R** também é relevante mas atua em um nicho mais reduzido de análise estatística). Fortran é o campeão de desempenho quando diz respeito a cálculos matemáticos, C/C++ já é um meio termo entre funcionalidades, desempenho e facilidade de programação, já python é o mais simples de se programar, porém com uma grande comunidade que cresce cada vez mais, e com isso possui muitas ferramentas poderosas. O python por si só não possui um desempenho competitivo com relação ao C e Fortran, porém, ele serve como uma ótima "cola" para bibliotecas/módulos feitos em C/C++ e Fortran fazendo com que códigos com alto desempenho possam ser programados muito mais rapidamente, isso significa menos tempo pensando em código e mais tempo focado na ciência!

Para um breve histórico sobre as primeiras linguagens de programação veja este texto e esse texto.

Sobre a evolução do uso de cada linguagem com o passar do tempo, incluindo a atual importância do python, veja esse vídeo.

# Ambiente para programação

Para este curso vamos usar o **Jupyter** como ambiente para programar. O **spyder** e o **pycharm** também são recomendados. Quando a tarefa for solicitada na forma de um notebook ela deve ser entregue em um **notebook .ipynb**, que pode ser gerado no Jupyter ou pycharm (no spyder também com pacotes adicionais). Quando a tarefa for um script isolado, para ser executado em terminal por exemplo, ela deve ser salva em um **arquivo .py**, que pode ser gerado com um editor do tipo gedit, notepad++ ou o spyder, também é possível usar o Jupyer e o pycharm para salvar como .py. Você pode também instalar o anaconda, que vai automaticamente instalar o Jupyter, além das bibliotecas/módulos mais utilizados. Porém para esse curso eu indico instalar separadamente as coisas usando **pip (o instalador de pacotes do python)** e **apt (utilitário para instalar pacotes do ubuntu e similares)**, sem o anaconda.

Tutoriais de instalação e funcionalidades:

- Jupyter (para ver um pouco das funcionalidades assista esse vídeo , os primeiros 3 minutos são sobre a instalação no windows)
- Spyder
- Pycharm
- Anaconda



# Estudo dirigido – Revisão de Python

Antes de vermos algumas bibliotecas a fundo vamos focar um pouco na revisão do básico de python, isto é, como funciona o **if**, **else**, **while**, **for**, os **tipos de variáveis básicas**, como definir **funções** , entre outras coisas.

Para a revisão dessa parte introdutória do python vocês devem estudar o conteúdo disponível nesse link (seções “Motivação” até “Entrada e saída de dados”), sobre o funcionamento básico da linguagem, nesse link (seção de “Introdução”) sobre algoritmos em Python. Assista também essa playlist para mais exemplos e alguns temas adicionais, além disso veja um pouco sobre list comprehension neste vídeo. Para as diferentes estruturas de dados (**listas, tuplas, arrays, dicionários**) veja os links: listas listas-vídeo tuplas tuplas-vídeo arrays (embora o tipo de array mais usado, e de nosso interesse, sejam os **arrays do numpy**, a ideia aqui é entender que em estrutura um array é imutável, isto é, sempre mantém suas dimensões iniciais e o tipo de variáveis que aloca, para os arrays do numpy veja o link e o vídeo numpy-vídeo-introdução, mais a frente veremos o array do numpy em detalhes), veja também esse link sobre dicionários dicionários-vídeo. Na prática, as estruturas mais usadas são arrays do numpy (por serem mais eficientes computacionalmente), em segundo as listas (pela flexibilidade), e menos comumente dicionários.

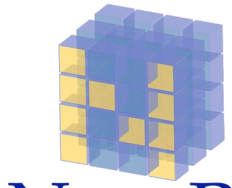
**Após esse estudo você deve ser capaz de:**

- **Definir condicionais e laços de repetição;**
- **Definir funções simples em python;**
- **Identificar e definir diferentes tipos de dados e estruturas de dados;**

# Bibliotecas importantes
# Estudo dirigido - Numpy e Math

Importante é pouco para o numpy, eu diria que **o numpy é essencial para quem programa qualquer coisa em python**. Em sua raiz o numpy é feito em C, o que o traz um grande desempenho. Tanto o numpy quanto o **math** (veja exemplos do math no link) possuem um rico arsenal de funções matemáticas, porém o numpy traz também ferramentas de manipulação de arrays, além de ser ter um desempenho muito superior. Então se uma função existir nos dois (numpy e math) 99,999...% das vezes será melhor usar o numpy.
Além disso, o numpy é largamente utilizado em data science. O math não tem a pretensão de ser tão poderoso, é apenas o pacote básico de matemática do python. Vamos focar então no numpy.

Assista essa playlist para estudar o numpy, desde o básico até algumas ferramentas avançadas. Na documentação do numpy você pode encontrar ainda mais funções interessantes, na dúvida consulte essa documentação (escreva a função desejada na barra de pesquisa "quick search").




```
import math
import numpy as np
```

**Após esse estudo você deve ser capaz de:**
- **Criar arrays com diferentes dimensões e realizar operações sobre eles;**
- **Entender o formato de indexação do numpy, fatiar e transpor matrizes (teste o comando array.T ou np.transpose(array));**
- **Entender o conceito de máscaras;**
- **Ter noções das funções disponíveis no numpy (dê uma olhada nessa parte da documentação também ).**

# Bibliotecas importantes
# Estudo dirigido – Pandas

A biblioteca pandas é muito útil para tratar dados mais complexos, com dimensões misturando tipos de dados diferentes, além de ser muito poderoso na hora de "limpar" os dados, remover linhas e colunas seguindo padrões e reorganizar as mesmas. O numpy capaz de fazer coisas similares, porém não tão poderoso quanto o pandas em termos de recursos para manipular estruturas de dados. Isto dito, o numpy é mais eficiente para operar com dados, o pandas tem mais recursos para manipular os dados. Caso seja possível fazer a mesma coisa com ambos, é praticamente certo que o numpy será mais veloz, por isso tente sempre usar o mesmo. Porém nem sempre os dados vêm limpos e perfeitos da forma que o numpy precisa. As vezes temos strings e números juntos, linhas com diferentes números de colunas, entre outros problemas que devemos remover com o pandas (para ter essa flexibilidade o pandas trabalha muito com listas e dicionários, perdendo desempenho).

Para estudar o pandas assista a essa playlist (da aula **"#06 – Introdução ao Pandas & Series"** até a aula **"#15 – Método Apply e Ordenaço de DataFrames"**).

**Após esse estudo você deve ser capaz de:**

- **Criar dataframes, filtrar dados, remover e criar colunas;**
- **Agrupar e tratar dados, juntar e ordenar dados;**
- **Ler um arquivo CSV;**

# Bibliotecas importantes
# Scipy e Astropy


As bibliotecas scipy e astropy possuem um grande conjunto de ferramentas. Scipy atua em um nicho mais amplo, possui funções de interpolação, minimização, geração de distribuições estatísticas, integração, solução de equações diferenciais e muito mais, sendo basicamente uma coletânea das mais importantes ferramentas matemáticas. Utilizaremos bastante essa biblioteca, porém apenas mais a frente no curso, por agora apenas dê uma olhada (só para ter noção das funcionalidades disponíveis) nos exemplos de funções básicas, integração, otimização, interpolação e álgebra linear na documentação (seção de tutoriais) do scipy. A biblioteca astropy é focada em ferramentas voltadas para a astronomia, como calculo de distâncias cosmológicas, ajustes de parâmetros astrofísicos e cosmológicos, ferramentas para trabalhar com espectroscopia e fotometria, cálculo de órbitas, tratamento de imagens astronômicas, mudanças de sistemas de coordenadas, acesso a importantes bancos de dados e muito mais, incluindo diversas ferramentas de estatística e interpolação. Veja alguns exemplos do uso do astropy na documentação (seção de tutoriais), leia até o tutorial “Coords 2” (não precisa reproduzir os exemplos, é apenas para ganhar uma noção, para ganhar confiança teste apenas alguns comandos específicos). Este tutorial em específico será bem útil para a lista de exercícios.


**Após esse estudo você deve ser capaz de:**

- **Importar as bibliotecas scipy e astropy;**
- **Entender as áreas em que cada biblioteca atua e ter noções das funcionalidades acessíveis em cada uma;**
- **(Para a lista de exercícios) Obter a posição de um objeto através do nome e realizar mudanças de coordenadas;**

# Bibliotecas importantes
# Argparse

O argparse  uma biblioteca que permite passar argumentos para um código python através do terminal, o que pode facilitar muito a automatização de processos através de scripts bash (mais na frente no curso, quando estudarmos métodos de paralelização no python, vamos ver como usar bash+argparse para fazer uma espécie de paralelização simplista) ou ser útil para uma execução rápida via terminal.

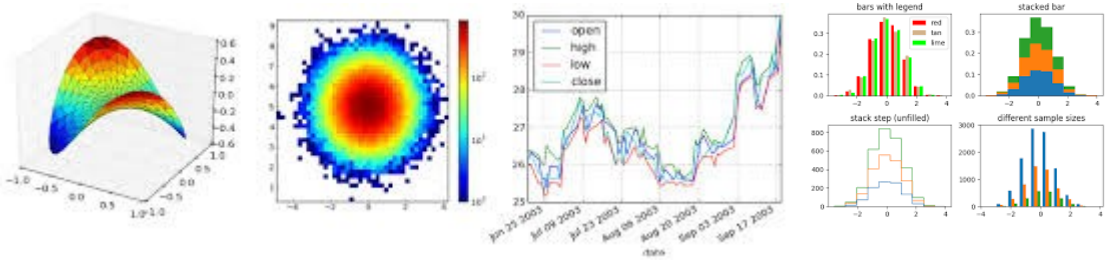
Para estudar a biblioteca argparse assista esse vídeo (ignorem temporariamente a linha de código ""if __name__='__main__':, o código funciona sem esta linha, mais a frente no curso vamos entender porque essa linha aparece no vídeo).  Veja também o tutorial da documentação neste link.

**Após esse estudo você deve ser capaz de:**

- **Criar um código simples que exija como entrada diferentes tipos de argumento via terminal.**


# Bibliotecas importantes
# Estudo dirigido – Matplotlib

Matplotlib é **a biblioteca mais utilizada para gerar gráficos**, embora a biblioteca seaborn tenha crescido bastante nos ultimo anos, sendo assim assim matplotlib é uma ferramenta essencial para qualquer astrônomo. Para esse tópico você deve estudar os seguintes links:Link introdução,link introdução 2,exemplos extras de como unir plots., como fazer histogramas, como fazer um gráfico polar, esses exemplos de como preencher entre plots preencher entre plots 2 preencher entre plots polares e como fazer plots em 3D nesse link.


**Após esse estudo você deve ser capaz de:**

- **Usar o matplotlib para gerar plots com pontos, linhas, histogramas e polar plots e plots 3D;**
- **Alterar nomes de eixos, título e incluir legendas;**
- **Juntar multiplos plots em uma única figura;**

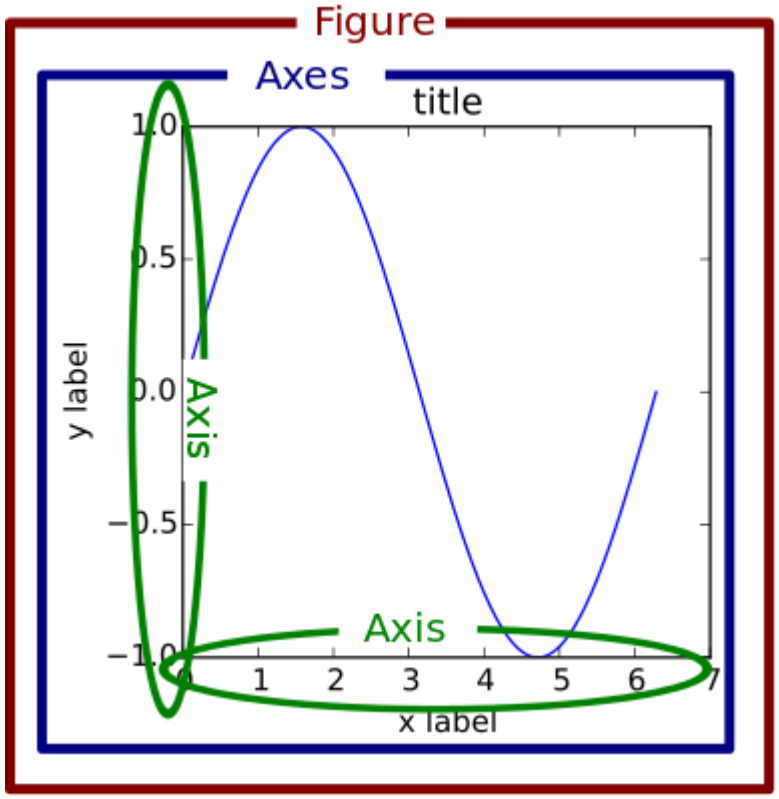
# Extra
# Matplotlib – Cores e Marcadores

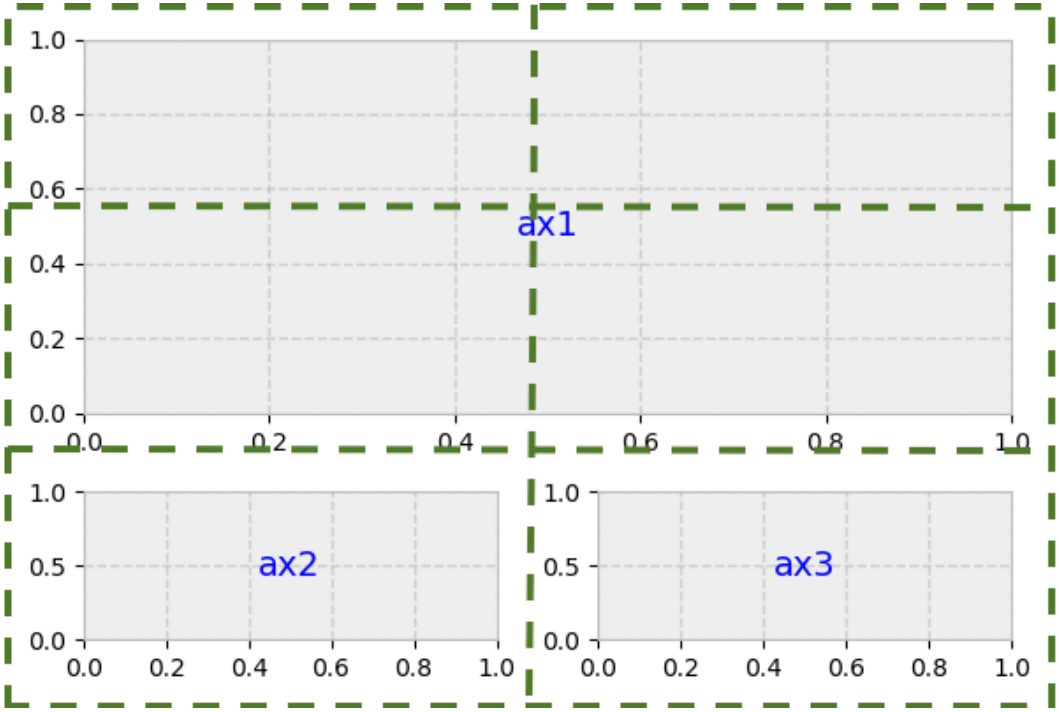
Pode ser útil saber as Cores com nomes definidos e marcadores básicos (marcadores são as figuras que vão representar os pontos do plot)

# Alguns tópicos avançados Matplotlib

Segue uma referência adicional com alguns tópicos avançados (alguns já vistos de outras formas nos links anteriores), como a hierarquia dos objetos do matplotlib e formas mais complexas de arranjo dos plots:

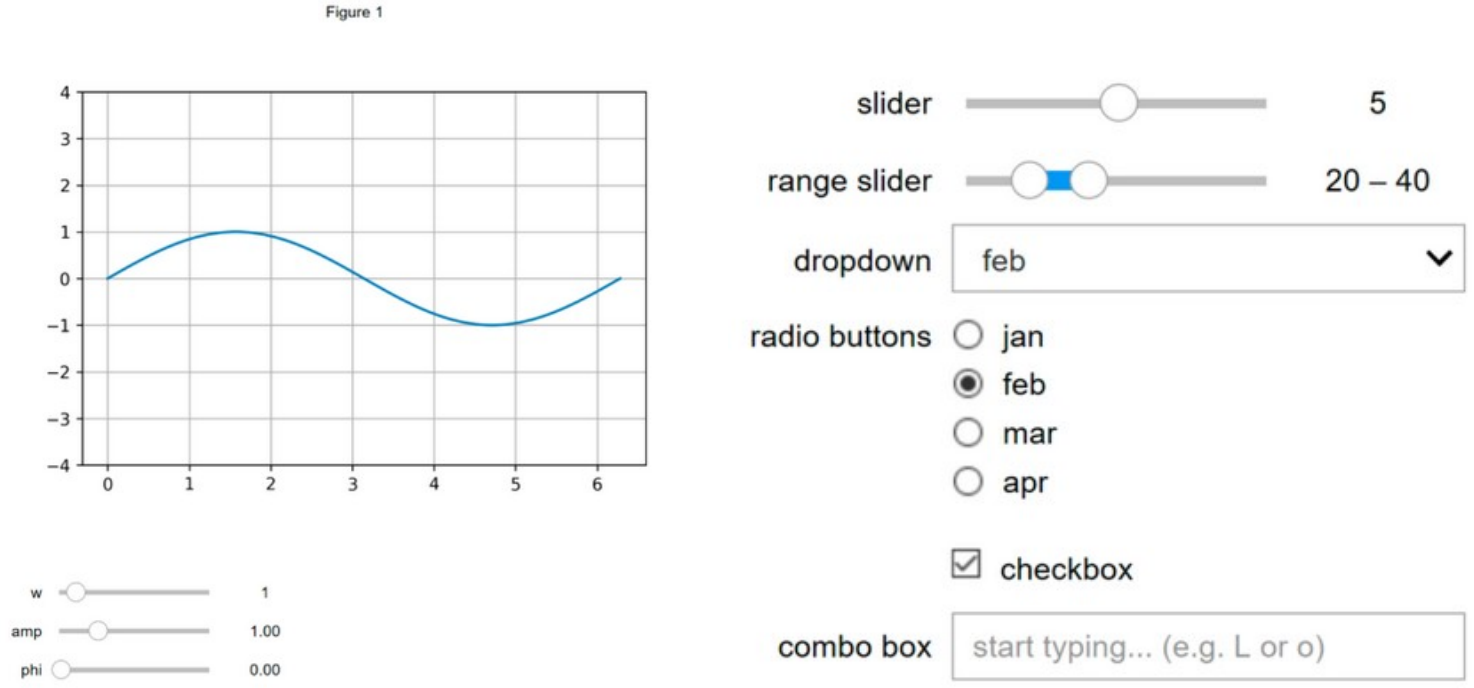
Link conceitos avançados no matplotlib



# ipyWidgets – plots interativos

Os widgets do jupyter criados com a biblioteca ipywidgets são bem interessantes quando procuramos um plot interativo, para “brincar com parâmetros” e demonstrar exemplos de possíveis comportamentos de uma função, principalmente em reuniões ou aulas. Neste link você pode estudar o básico do widgets interativos, focado no uso com o matplotlib.




**Após esse estudo você deve ser capaz de:**

- **Gerar plots interativos básicos (com sliders, dropdow list e botões);**

# FIM