

Introdução: **Anaconda e Notebook Jupyter.**

# Anaconda

O Anaconda é uma distribuição de pacotes construída para análise de dados. Embutido nela vem o conda, um gerenciador de pacotes e ambientes. Você usará o conda para criar ambientes que isolam projetos que usam versões e pacotes diferentes do Python. Você também o usará para instalar, desinstalar e atualizar pacotes nos diversos ambientes. O Anaconda é, na verdade, a distribuição de software que contém o conda, o Python e mais de 150 pacotes científicos e suas dependências.

# Anaconda

## Gerenciando pacotes

O conda instala pacotes pré-compilados. Por exemplo, a distribuição Anaconda contém as bibliotecas Numpy, Scipy e Scikit-learn compiladas com a [biblioteca MKL](#), o que acelera diversas operações matemáticas. O conda é parecido com o pip, exceto pelo fato de que os pacotes disponíveis são mais focados em *data science*, enquanto o pip é mais generalista.

## Ambientes

Ambientes permitem que você separe e isole pacotes que estão sendo utilizados para projetos diferentes. Códigos que dependem de versões diferentes de uma mesma biblioteca serão utilizados com frequência.

# Anaconda

## **Ambientes (exemplos de utilização)**

É possível trabalhar com código antigo que não funciona em Python 3 ou código novo que não roda em Python 2. Ter ambas as versões instaladas evita muita confusão e bugs. É muito melhor ter ambientes separados.

Você também pode exportar a lista de pacotes de um ambiente para um arquivo e, então, incluir esse arquivo no código. Isso permite que outras pessoas consigam instalar com facilidade todas as dependências para seu código rodar.

# Anaconda

## Instalação

O Anaconda está disponível para Windows, Mac OS X e Linux. É possível encontrar os instaladores e as instruções de instalação no site <https://www.continuum.io>

Caso já tenha o Python instalado em seu computador, essa instalação não estragará nada. Em vez disso, o Python padrão usado nos scripts e programas passará a ser o que vem dentro da distribuição Anaconda.

**Escolha a versão com o Python 3.6.**

# Anaconda

## No Windows

Diversas aplicações são instaladas junto ao Anaconda:

- **Anaconda Navigator**, uma interface para gerenciar seus pacotes e ambientes
- **Anaconda Prompt**, um terminal para usar a interface de linha de comando para gerenciar seus ambientes e pacotes
- **Spyder**, uma interface equipada para o desenvolvimento científico

Para evitar problemas futuros, é melhor atualizar todos os pacotes do ambiente padrão. Abra o programa **Anaconda**

**Prompt**, digite os seguintes comandos:

```
conda upgrade conda
```

```
conda upgrade --all
```

# Anaconda

## Gerenciando pacotes

Uma vez instalado o Anaconda, gerenciar pacotes é bastante simples. Para instalar um pacote, digite `conda install nome_do_pacote` no terminal. Por exemplo, para instalar o numpy, digite **`conda install numpy`**.

É possível instalar diversos pacotes ao mesmo tempo. Ao digitar algo como `conda install numpy scipy pandas`, todos os pacotes serão instalados simultaneamente. Também é possível especificar qual versão de um pacote você quer ao adicionar o número da versão, tal como em `conda install numpy=1.10`.

O conda também instala automaticamente as dependências para você. Por exemplo, o pacote `scipy` depende do pacote `numpy`, ele usa e exige o `numpy`. Caso você instale apenas o `scipy` (`conda install scipy`), o Conda também instalará o `numpy`, caso ainda não esteja instalado.

# Anaconda

## Gerenciando pacotes

A maioria dos comandos é intuitiva. Para remover, use `conda remove nome_do_pacote`. Para atualizar um pacote, **conda update nome\_do\_pacote**. Caso você queira atualizar todos os pacotes de um ambiente, o que pode ser bem útil, use `conda update --all`. Por fim, para listar os pacotes instalados, use o comando **conda list** que vimos anteriormente.

A maioria dos comandos é intuitiva. Para remover, use `conda remove nome_do_pacote`. Para atualizar um pacote, `conda update nome_do_pacote`. Caso você queira atualizar todos os pacotes de um ambiente, o que pode ser bem útil, use `conda update --all`. Por fim, para listar os pacotes instalados, use o comando `conda list` que vimos anteriormente.

Caso você não saiba exatamente o nome do pacote que está buscando, é possível tentar encontrá-lo com o comando `conda search termo_de_busca`.



# Anaconda

## Gerenciando ambientes

o conda pode ser utilizado para criar ambientes que isolem seus projetos. Para criar um ambiente, use `conda create -n nome_amb lista de pacotes` no terminal. Aqui, `-n nome_amb` dá o nome do seu ambiente (`-n` de nome), e `lista de pacotes` é a lista de pacotes que você quer instalada no ambiente. Por exemplo, para criar um ambiente chamado `my_env` e instalar o numpy nele, digite `conda create -n my_env numpy`.

Ao criar um ambiente, você pode especificar qual versão do Python quer que seja instalada nele. Isso é útil para quando você trabalha com código tanto em Python 2.x como em Python 3.x. Para criar um ambiente com uma versão específica de Python, faça algo como `conda create -n py3 python=3` ou `conda create -n py2 python=2`.

Esses comandos instalarão as versões mais recentes de Python 3 e Python 2, respectivamente. Para instalar uma versão específica, use, por exemplo, `conda create -n py python=3.3` para a versão 3.3 do Python.

# Anaconda

## Entrando em um ambiente

Uma vez criado o ambiente, use o comando `source activate my_env` para entrar nele no OSX/Linux. No Windows, use `activate my_env`.

O ambiente tem apenas alguns pacotes instalados automaticamente, além daqueles inseridos no comando de criação. É possível checar isso usando o comando `conda list`. Instalar pacotes no ambiente é feito da mesma maneira: `conda install nome_do_pacote`, apenas com uma diferença: desta vez, os pacotes específicos que forem instalados estarão disponíveis apenas enquanto o ambiente estiver ativo. Para sair do ambiente, digite `source deactivate` (no OSX/Linux). No Windows, use `deactivate`.

Qual comando você usaria para criar um ambiente chamado `data` com Python 3.6, Numpy e Pandas instalados?

# Anaconda

## Salvando e carregando ambientes

Uma ótima característica de compartilhar ambientes é que outras pessoas podem instalar todos os pacotes utilizados em seu código com a versão correta. Você pode salvar os pacotes em um arquivo [YAML](#) usando o comando `conda env export > environment.yaml`. A primeira parte, `conda env export`, escreve todos os pacotes no ambiente, incluindo a versão Python.

É possível ver, acima, o nome do ambiente e todas as dependências (assim como suas versões). A segunda parte do comando de exportar, `> environment.yaml`, escreve o texto exportado em um arquivo YAML chamado `environment.yaml`. Esse arquivo pode, então, ser compartilhado com outros para criar o mesmo ambiente usado em seu projeto.

Para criar um ambiente de um arquivo de ambiente, use o comando `conda env create -f environment.yaml`. Isso criará um novo ambiente com o mesmo nome contido no arquivo `environment.yaml`.

# Anaconda

## Compartilhando ambientes

Ao compartilhar seu código no GitHub, é uma boa prática fazer um arquivo de ambiente e inclui-lo no repositório. Isso facilitará a instalação de todas as dependências de seu código pelos usuários. Eu sempre incluo também um arquivo `requirements.txt` usando o `pip freeze` ([saiba mais aqui](#)) para quem não estiver usando o conda.

## Aprendendo mais

Para aprender mais sobre o conda e como ele se encaixa no ecossistema Python, dê uma olhada nesse artigo de Jake Vanderplas: [Conda myths and misconceptions](#)(em tradução livre, "Conda: Mitos e Enganos").

# Anaconda

## Recomendações

### Usando ambientes

Algo que me ajudou demais foi ter ambientes separados para o Python 2 e Python 3. Eu usei os comandos `conda create -n py2 python=2` e `conda create -n py3 python=3` para criar dois ambientes separados, `py2` e `py3`.

Agora, tenho um ambiente de uso geral para cada uma das versões de Python. Em cada um deles, instalei a maioria dos pacotes padrão de *data science* (Numpy, Scipy, Pandas, etc.).

Também achei útil criar ambientes para cada projeto em que estou trabalhando.

# Anaconda

## Listando ambientes

Caso esqueça os nomes dos ambientes (acontece comigo às vezes), use o comando `conda env list` para listar todos os ambientes criados por você. Deveria aparecer uma lista de ambientes, assim como um asterisco ao lado do ambiente ativo. O ambiente padrão, aquele usado quando nenhum outro está ativo, é chamado de `root`.

## Removendo ambientes

Caso existam ambientes que não são mais úteis, `conda env remove -n nome_amb` é o comando que remove o ambiente escolhido (no caso, o nomeado `nome_amb`).

# Notebooks Jupyter

## O que são os notebooks Jupyter?

O notebook [Jupyter](#) é uma aplicação web que permite que você combine texto explicativo, equações matemáticas, código e visualizações em um único documento facilmente compartilhável.

Os notebooks também são carregados automaticamente no GitHub. Essa é uma característica muito poderosa na hora de compartilhar o trabalho feito. Existe também o site <http://nbviewer.jupyter.org/>, que carrega os notebooks de um repositório Github ou de qualquer outro lugar.

# Notebooks Jupyter

## Programação literária

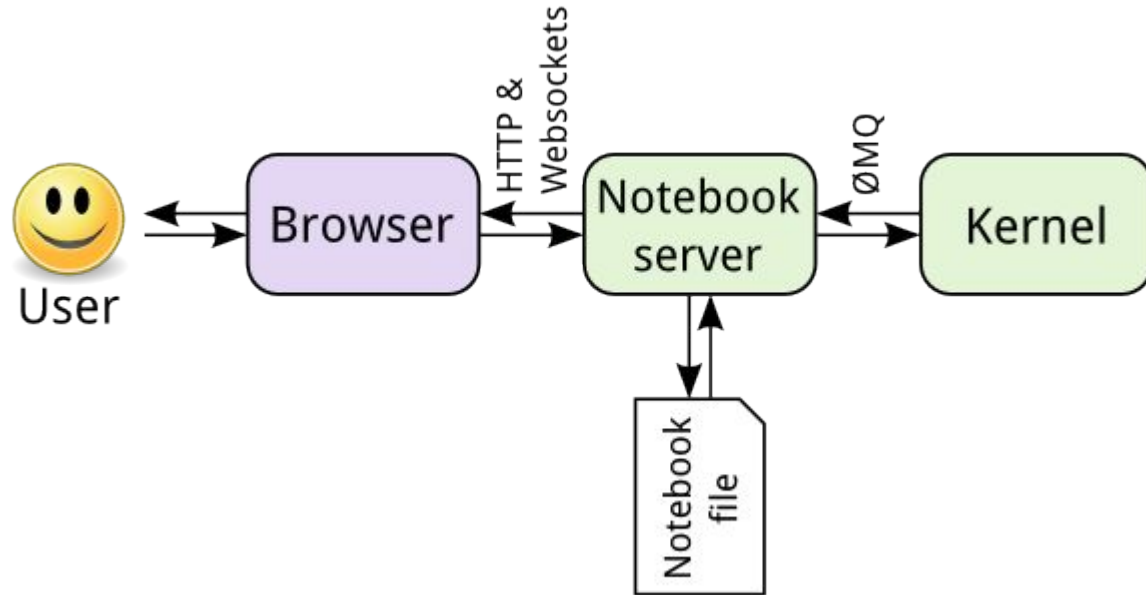
Em vez de pensar que nossa tarefa principal é dizer ao computador o que fazer, vamos nos concentrar em explicar para outros seres humanos o que queremos que o computador faça. “Donald Knuth”

## Como os notebooks funcionam

Os notebooks Jupyter surgiram a partir do [projeto IPython](#), iniciado por Fernando Perez. IPython é um terminal interativo, similar ao terminal Python normal, mas com recursos ótimos como destaques de sintaxe e autopreenchimento para código. Originalmente, os notebooks funcionavam ao mandar mensagens do aplicativo web (o notebook que você visualize no navegador) para um núcleo IPython (uma aplicação IPython que rodava em segundo plano). O núcleo executava o código e, então, mandava-o de volta para o notebook.



# Notebooks Jupyter



# Notebooks Jupyter

## Instalando o notebook Jupyter

O jeito mais fácil de instalar o Jupyter é claramente baixando o Anaconda. Os notebooks Jupyter vêm embutidos na distribuição. É possível usar os notebooks já no ambiente padrão.

Para instalar os notebooks Jupyter em um ambiente do conda, use `conda install jupyter notebook`.

# Notebooks Jupyter

## Lançando o servidor do notebook

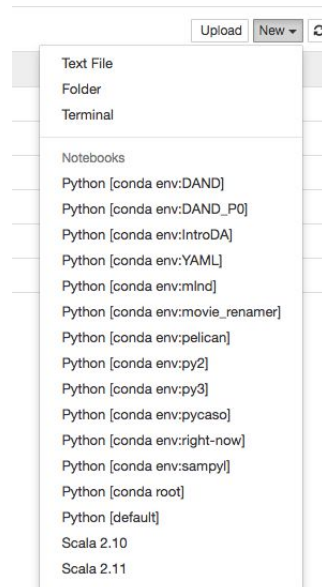
Para lançar um servidor notebook, digite `jupyter notebook` no terminal ou prompt. Isso inicializará o servidor no diretório em que você lançou o comando. Isso significa que os arquivos do notebook serão salvos neste diretório. A prática padrão é inicializar o servidor no diretório onde os notebooks se encontram. No entanto, é possível navegar pelo sistema de arquivos para o local onde se encontram os seus notebooks.

Ao rodar esse comando (faça uma tentativa!), a página inicial do servidor deve se abrir em seu navegador. Na definição padrão, o notebook roda no endereço `http://localhost:8888`. Caso não esteja familiarizado com isso, `localhost` significa o seu computador e `8888` é a porta que o servidor está usando. Enquanto o servidor estiver rodando, sempre será possível voltar para ele ao digitar <http://localhost:8888> em seu navegador.

# Notebooks Jupyter

No canto direito superior, você pode clicar em "New" para criar um notebook, arquivo de texto, pasta ou terminal novo. A lista abaixo de "notebooks" mostra os núcleos (kernels) que você tem instalados.

Caso você rode o servidor do notebook Jupyter de um ambiente conda, também será possível escolher os núcleos de quaisquer outros ambientes. Para criar um notebook novo, clique no núcleo que deseja usar.

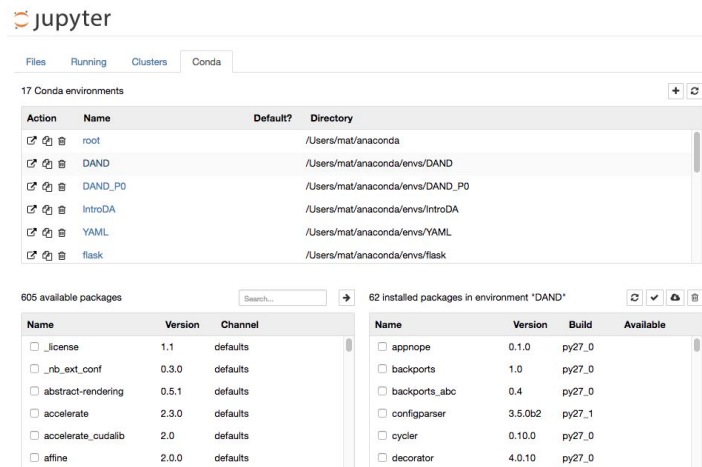


# Notebooks Jupyter

As abas no topo mostram *Files*, *Running* e *Cluster*. *Files* mostra todos os arquivos e pastas do diretório atual. Clicar na aba *Running* listará todos os notebooks atualmente ativos. Neste ponto, é possível gerenciá-los.

*Clusters* era onde antes você podia criar núcleos múltiplos para usar em computação paralela. Agora, isso foi tomado pelo [ipyparallel](#), então, não há nada demais a ser feito aqui.

Caso esteja rodando o servidor do notebook de um ambiente conda, você também terá acesso a uma aba nomeada "**Conda**", como mostraremos abaixo. Aqui, é possível administrar os ambientes de dentro do Jupyter. É possível criar ambientes, instalar pacotes, atualizar pacotes, exportar ambientes e muito mais.



The screenshot shows the Jupyter web interface with the 'Conda' tab selected. The interface is divided into two main sections: '17 Conda environments' and '62 installed packages in environment "DAND"'. The '17 Conda environments' section contains a table with columns 'Action', 'Name', 'Default?', and 'Directory'. The '62 installed packages in environment "DAND"' section contains a table with columns 'Name', 'Version', 'Build', and 'Available'.

Action	Name	Default?	Directory
	root		/Users/mat/anaconda
	DAND		/Users/mat/anaconda/envs/DAND
	DAND_P0		/Users/mat/anaconda/envs/DAND_P0
	IntroDA		/Users/mat/anaconda/envs/IntroDA
	YAML		/Users/mat/anaconda/envs/YAML
	flask		/Users/mat/anaconda/envs/flask

Name	Version	Channel
<input type="checkbox"/> _license	1.1	defaults
<input type="checkbox"/> _nb_ext_conf	0.3.0	defaults
<input type="checkbox"/> abstract-rendering	0.5.1	defaults
<input type="checkbox"/> accelerate	2.3.0	defaults
<input type="checkbox"/> accelerate_cudatib	2.0	defaults
<input type="checkbox"/> affine	2.0.0	defaults

Name	Version	Build	Available
<input type="checkbox"/> appnope	0.1.0	py27_0	
<input type="checkbox"/> backports	1.0	py27_0	
<input type="checkbox"/> backports_abc	0.4	py27_0	
<input type="checkbox"/> configparser	3.5.0b2	py27_1	
<input type="checkbox"/> cyder	0.10.0	py27_0	
<input type="checkbox"/> decorator	4.0.10	py27_0	

# Notebooks Jupyter

## Desligando o Jupyter

É possível desligar os notebooks individualmente ao marcar as caixas ao lado de cada notebook na página inicial do servidor e, depois, clicar em "Shutdown". É bom garantir que todas as mudanças tenham sido salvas antes de fazer isso! Quaisquer mudanças feitas desde a última vez que o arquivo foi salvo serão perdidas. Também será necessário rodar novamente todos os códigos escritos da próxima vez que iniciar o notebook.

Também é possível desligar o servidor inteiro ao pressionar as teclas control + C duas vezes no terminal. De novo, isso desligará imediatamente todos os notebooks.

# Notebooks Jupyter

## Células de código

A maioria do seu trabalho nos notebooks será feita em células de código. É nelas que você escreve e executa o código. Nessas células, é possível escrever qualquer tipo de código, declarando variáveis, definindo funções e classes, importando pacotes e muito mais. Qualquer código executado em uma célula fica disponível para todas as outras.

Baixe o notebook abaixo, chamado **Working With Code Cells (Google Classroom)**, e então rode-o em seu próprio servidor (no terminal, mude para o diretório com o arquivo do notebook e, então, digite o comando `jupyter notebook`). O navegador talvez já tente abrir o arquivo do notebook sem sequer tê-lo baixado. Caso isso aconteça, clique no link com o botão direito e escolha a opção "Salvar link como..."

# Notebooks Jupyter

## Células de código

A maioria do seu trabalho nos notebooks será feita em células de código. É nelas que você escreve e executa o código. Nessas células, é possível escrever qualquer tipo de código, declarando variáveis, definindo funções e classes, importando pacotes e muito mais. Qualquer código executado em uma célula fica disponível para todas as outras.

Baixe o notebook abaixo, chamado **Working With Code Cells (Google Classroom)**, e então rode-o em seu próprio servidor (no terminal, mude para o diretório com o arquivo do notebook e, então, digite o comando `jupyter notebook`). O navegador talvez já tente abrir o arquivo do notebook sem sequer tê-lo baixado. Caso isso aconteça, clique no link com o botão direito e escolha a opção "Salvar link como..."



# Notebooks Jupyter

## Células markdown

Como mencionado anteriormente, as células também podem ser usadas para texto escrito em markdown. Markdown é uma sintaxe de formatação que permite a inclusão de links, textos estilizados como negrito ou itálico, assim como código formatado. Assim como nas células de código, ao apertar **Shift + Enter** ou **Control + Enter** para rodar a célula de markdown, o lugar onde ela está carregará o Markdown como texto formatado. Incluir textos permite que você escreva uma narrativa junto a seu código, assim como registrar o que estava passando pela sua cabeça no momento que escreveu o código e também documentá-lo.

É possível encontrar a [documentação aqui](#)

# Notebooks Jupyter

## Cabeçalhos

É possível escrever cabeçalhos usando o símbolo jogo da velha # antes do texto. Um # gera um cabeçalho h1, dois #s geram um h2 e assim por diante. Isso se dá da seguinte maneira:

```
# Cabeçalho 1
```

```
## Cabeçalho 2
```

```
### Cabeçalho 3
```

O código acima gera **Cabeçalho 1**      **Cabeçalho 2**      **Cabeçalho 3**

# Notebooks Jupyter

## Links

Colocar links em Markdown é feito ao cercar o texto em colchetes e, então, a URL em parênteses, desta forma: [Página do curso:](<http://cc.uffs.edu.br>) para gerar um link.

## Ênfase

É possível enfatizar o texto usando negrito ou itálico com asteriscos ou sublinhados (\* ou \_). Para itálico, inicie e termine o texto com um asterisco ou sublinhado, `_gelato_` ou `*gelato*` geram *gelato*.

Texto em negrito usa dois símbolos, `**abacate**` ou `__abacate__` geram **abacate**.

Tanto asteriscos como sublinhados funcionam, desde que sejam usados os mesmos símbolos de ambos os lados do texto em questão.

# Notebooks Jupyter

## Código

Existem duas maneiras diferentes de mostrar código, dentro do texto ou como um bloco de código separado do texto. Para o formato dentro do texto, insira acentos graves antes e depois do código. Por exemplo, ``string.punctuation`` gera `string.punctuation`.

Para criar um bloco de código, comece uma nova linha e delimite o código com três acentos graves:

```
'''
```

```
import requests
```

```
response = requests.get('http://cc.uffs.edu.br')
```

```
'''
```

Ou então faça uma indentação de cada linha do código com quatro espaços.

```
import requests
```

```
response = requests.get('https://www.udacity.com')
```

# Notebooks Jupyter

## Expressões matemáticas

É possível criar expressões matemáticas em células markdown usando os símbolos do [LaTeX](#). Os notebooks usam MathJax para carregar os símbolos do LaTeX como símbolos matemáticos. Para entrar no modo matemático, envolva o LaTeX em cifrões `$y = mx + b$` para matemática dentro do texto. Para um bloco de expressões, use cifrões duplos:

`$$`

`y = \frac{a}{b+c}`

`$$`

[resumo](#)

# Notebooks Jupyter

## Atalhos do teclado

Os notebooks vêm com diversos atalhos do teclado que permitem que você use o teclado para interagir com as células, em vez de usar o mouse e a barra de ferramentas. Eles exigem algum tempo de aprendizado, mas, quando você fica confiante no uso desse recurso, o fluxo de trabalho fica muito mais rápido. Para aprender mais a respeito dos atalhos e praticar o uso deles, baixe o notebook **Atalhos do teclado** abaixo. De novo, é possível que o navegador tente abri-lo, mas você deve salvá-lo no computador. Clique com o botão direito no link e escolha "Salvar link como..."

Material de apoio

[Keyboard Shortcuts](#)

# Notebooks Jupyter

## Palavras-chave mágicas

Palavras-chave mágicas são comandos especiais que você pode rodar em células que permitem que controle o notebook ou então execute comandos do sistema, tais como mudar de diretório. Por exemplo, você pode fazer a biblioteca `matplotlib` funcionar interativamente com o notebook ao digitar `%matplotlib`.

Os comandos mágicos são precedidos por um ou dois símbolos de porcentagem (`%` ou `%%`) para mágicas de linha ou de célula, respectivamente. As mágicas de linha se aplicam apenas na linha em que o comando foi escrito, as de célula, na célula inteira.

# Notebooks Jupyter

**Observação:** essas palavras-chave mágicas são específicas do kernel de Python normal. Caso você esteja usando outro núcleo, elas provavelmente não funcionarão.

```
In [21]: from math import sqrt

def fibol(n): # Recursive Fibonacci number
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fibol(n-1) + fibol(n-2)

def fibo2(n): # Closed form
    return ((1+sqrt(5))**n-(1-sqrt(5))**n)/(2**n*sqrt(5))
```

```
In [22]: %timeit fibol(20)

100 loops, best of 3: 3.49 ms per loop
```

```
In [23]: %timeit fibo2(20)

The slowest run took 16.75 times longer than the fastest. This could mean
that an intermediate result is being cached.
1000000 loops, best of 3: 1.08 µs per loop
```



# Notebooks Jupyter

Caso queira saber quanto tempo uma célula inteira demorou para rodar, use o comando `%%timeit` da seguinte maneira:

```
In [24]: import random
```

```
In [97]: %%timeit
prize = 0
for ii in range(100):
    # roll a die
    roll = random.randint(1, 6)
    if roll%2 == 0:
        prize += roll
    else:
        prize -= 1
```

10000 loops, best of 3: 148  $\mu$ s per loop

```
In [98]: %%timeit
rolls = (random.randint(1,6) for _ in range(100))
prize = sum(roll if roll%2 == 0 else -1 for roll in rolls)
```

10000 loops, best of 3: 154  $\mu$ s per loop

# Notebooks Jupyter

## Embutindo visualizações em notebooks

Como mencionado anteriormente, os notebooks permitem que você inclua imagens ao longo do texto e código. Isso é muito útil, especialmente quando se usa o `matplotlib` ou outros pacotes de visualização para criar gráficos e imagens. É possível usar o comando `%matplotlib` para carregar o pacote `matplotlib` de modo interativo no notebook. O padrão é carregar as imagens em uma janela própria. No entanto, é possível passar um argumento para o comando de modo que ele selecione um ["backend"](#) específico, o software que carrega a imagem. Para carregar as imagens diretamente no notebook, é preciso usar o comando de *backend inline*: `%matplotlib inline`.

***Dica:** em telas de resolução elevada, tais como as com display Retina, as imagens padrão do notebook podem ficar borradas.*

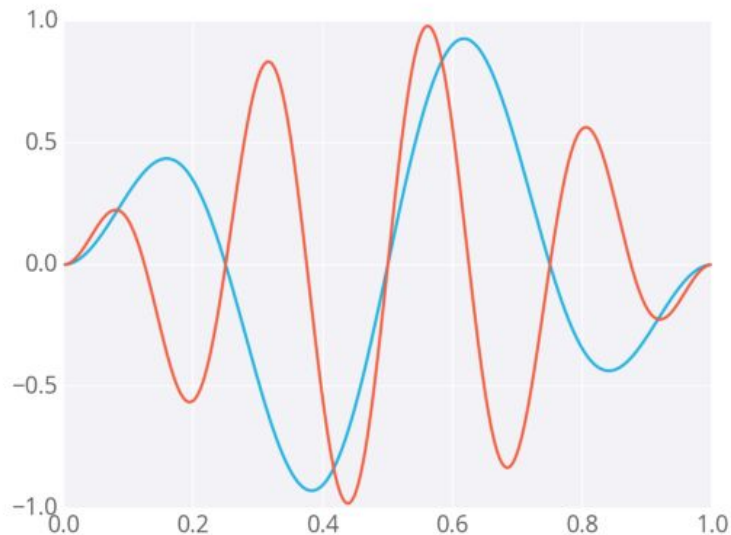
*Use o comando `%config InlineBackend.figure_format = 'retina'` depois do `%matplotlib inline` para processar imagens de alta resolução.*

# Notebooks Jupyter

```
In [103]: %matplotlib inline
          %config InlineBackend.figure_format = 'retina'

          import matplotlib.pyplot as plt
          import numpy as np
```

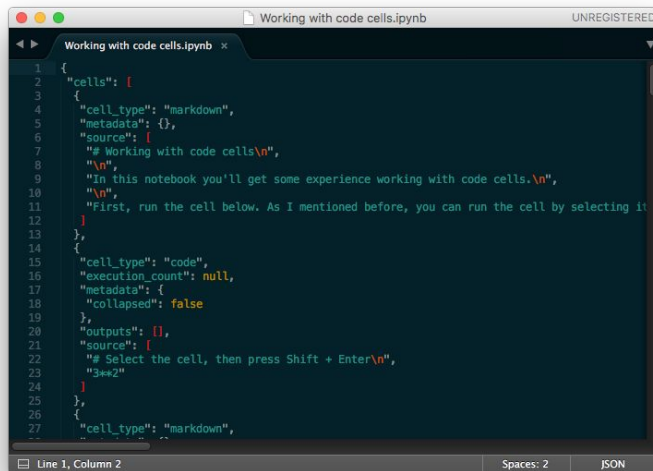
```
In [134]: x = np.linspace(0, 1, 300)
          for w in range(2, 6, 2):
              plt.plot(x, np.sin(np.pi*x)*np.sin(2*w*np.pi*x))
```



# Notebooks Jupyter

## Convertendo notebooks

Os notebooks são grandes arquivos [JSON](#) com a extensão `.ipynb`.



The screenshot shows a Jupyter Notebook editor window titled "Working with code cells.ipynb". The editor displays the JSON structure of the notebook file. The JSON is a dictionary with a "cells" key, which is a list of cell objects. The first cell is a markdown cell containing introductory text. The second cell is a code cell containing instructions on how to run a cell. The third cell is a markdown cell. The editor has a dark theme and a status bar at the bottom showing "Line 1, Column 2", "Spaces: 2", and "JSON".

```
1 {
2   "cells": [
3     {
4       "cell_type": "markdown",
5       "metadata": {},
6       "source": [
7         "# Working with code cells\n",
8         "\n",
9         "In this notebook you'll get some experience working with code cells.\n",
10        "\n",
11        "First, run the cell below. As I mentioned before, you can run the cell by selecting it
12      ]
13    },
14    {
15      "cell_type": "code",
16      "execution_count": null,
17      "metadata": {
18        "collapsed": false
19      },
20      "outputs": [],
21      "source": [
22        "# Select the cell, then press Shift + Enter\n",
23        "3*3=2"
24      ]
25    },
26    {
27      "cell_type": "markdown",
28      "source": [
29        "# Working with code cells\n",
30        "\n",
31        "In this notebook you'll get some experience working with code cells.\n",
32        "\n",
33        "First, run the cell below. As I mentioned before, you can run the cell by selecting it
34      ]
35    }
36  ]
37 }
```

# Notebooks Jupyter

## Convertendo notebooks

Como os notebooks são JSON, é simples de convertê-los para outros formatos. O Jupyter vem com um recurso chamado `nbconvert` para converter para HTML, markdown, slideshows etc...

Por exemplo, para converter um notebook em um arquivo HTML, digite no terminal

```
jupyter nbconvert --to html notebook.ipynb
```

Converter para HTML é útil para compartilhar seus notebooks com pessoas que não usam essa ferramenta. Markdown é ótimo para incluir o notebook em blogs e outros editores de texto que aceitam o formato markdown.

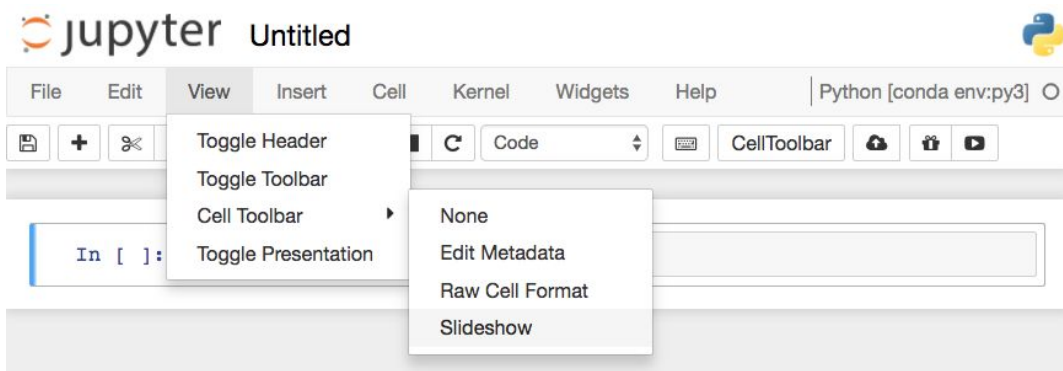
Como sempre, é possível aprender mais sobre o `nbconvert` na [documentação](#).

# Notebooks Jupyter

## Criando uma apresentação

Criar uma apresentação a partir de um notebook é um dos meus recursos favoritos. [Aqui, você pode ver um exemplo de apresentação](#) introduzindo o Pandas para trabalhar com dados.

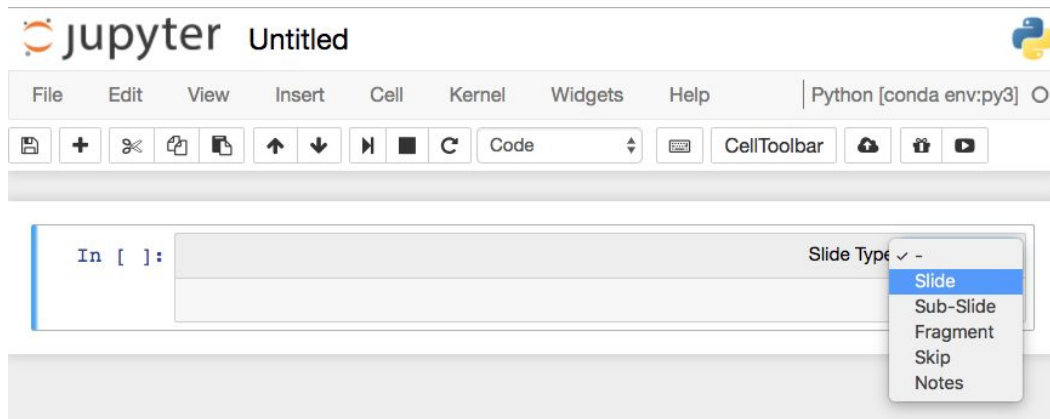
Os slides são criados no notebook como células normais, mas será necessário designar quais células são slides e o tipo de slide que a célula será. Na barra de menu, clique em *View > Cell Toolbar > Slideshow* para abrir o menu de slide de cada célula.



# Notebooks Jupyter

## Criando uma apresentação

Isso mostrará um menu em cada célula, que permite escolher como a célula aparecerá na apresentação.



# Notebooks Jupyter

## Criando uma apresentação

**Slides** são slides completos que são passados da esquerda para a direita. **Sub-slides** aparecem na apresentação ao pressionar para cima ou para baixo. **Fragments** começam ocultos, então, aparecem ao apertar um botão. É possível pular células da apresentação com **Skip**, e **Notes** deixa aquela célula como anotações do autor.



# Notebooks Jupyter

## Rodando a apresentação

Para criar a apresentação a partir do arquivo notebook, será necessário usar o `nbconvert`:

```
jupyter nbconvert notebook.ipynb --to slides
```

Isso converte o notebook para os arquivos necessários para a apresentação, mas será necessário fornecer para ele um servidor HTTP para visualizar a apresentação.

Para converter e ver imediatamente, use

```
jupyter nbconvert notebook.ipynb --to slides --post serve
```