

MARCO ANTONIO LEONEL CAETANO

IMERSÃO EM PYTHON

**Lições básicas para iniciantes
em programação de computadores**



Blucher

Marco Antonio Leonel Caetano

IMERSÃO EM PYTHON

Lições básicas para iniciantes em
programação de computadores

Imersão em Python: lições básicas para iniciantes em programação de computadores

© 2026 Marco Antonio Leonel Caetano

Editora Edgard Blücher Ltda.

Publisher Edgard Blücher

Editor Eduardo Blücher

Coordenação editorial Rafael Fulanetti

Coordenação de produção Ana Cristina Garcia

Produção editorial Marcel Iha

Preparação de texto Maurício Katayama

Diagramação Guilherme Salvador

Revisão de texto Alexandra Fonseca

Capa Juliana Midori Horie

Imagem da capa iStockphoto

Blucher

Rua Pedroso Alvarenga, 1245, 4º andar

04531-934 – São Paulo – SP – Brasil

Tel.: 55 11 3078-5366

contato@blucher.com.br

www.blucher.com.br

Segundo o Novo Acordo Ortográfico, conforme 6. ed. do Vocabulário Ortográfico da Língua Portuguesa, Academia Brasileira de Letras, julho de 2021.

É proibida a reprodução total ou parcial por quaisquer meios sem autorização escrita da editora.

Todos os direitos reservados pela Editora Edgard Blücher Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)
Heytor Diniz Teixeira, CRB-8/10570

Caetano, Marco Antonio Leonel

Imersão em Python : lições básicas para iniciantes em programação de computadores / Marco Antonio Leonel Caetano. – São Paulo : Blucher, 2026.

270 p. : il.

Bibliografia

ISBN 978-85-212-3371-8 (Impresso)

ISBN 978-85-212-3369-5 (Eletrônico – Epub)

ISBN 978-85-212-3368-8 (Eletrônico – PDF)

1. Linguagem de programação – Python. 2. Linguagens de computador. 3. Linguagem de programação – Estudo e ensino. 4. Python – Estudo e ensino. 5. Programação de computadores. I. Título.

CDU 004.43

Índice para catálogo sistemático:
1. Linguagem de programação CDU 004.43

CONTEÚDO

INTRODUÇÃO.....	13
1. AMBIENTE DE PROGRAMAÇÃO	17
1.1 Escolhendo o ambiente de programação.....	17
1.2 Ambiente de colaboradores do Google (Colab)	18
1.3 Ambiente Jupyter Anaconda	19
1.4 Primeiros programas em Jupyter	22
1.5 Alterando e editando células no Jupyter	24
1.6 Utilização do Kernel.....	27
1.7 Salvar e download	29
2. CRIAÇÃO E MANIPULAÇÃO DE DADOS EM LISTAS	33
2.1 Criando uma lista.....	33
2.2 Funções da matemática	42
2.3 Funções estatísticas.....	44
2.4 Exemplos de operações com listas	45
2.5 Exercícios para retenção de conhecimento.....	50

3. TRABALHANDO COM VETORES EM ARRAY	57
3.1 O que é um array?	57
3.2 Trabalhando com a Numpy	58
3.3 Gerando sequências de números automáticos na Numpy.....	60
3.4 Operações algébricas e estatísticas com array	62
3.5 Gráficos com array.....	67
3.6 Múltiplos gráficos em subplots	72
3.7 Exercícios para retenção de conhecimento.....	74
4. A MATEMÁTICA E O PYTHON	77
4.1 Operações elementares	77
4.2 Variáveis	80
4.3 Tipos de variáveis em Python.....	82
4.4 Funções matemáticas.....	84
4.5 Soluções de sistemas lineares	90
4.6 Cálculo de áreas de funções.....	99
4.7 Exercícios para retenção de conhecimento.....	104
5. PROBABILIDADE E ESTATÍSTICA COM PYTHON	109
5.1 Estatística descritiva	109
5.2 Gráficos da estatística em Python	114
5.3 As probabilidades	121
5.3.1 Calculando probabilidade com densidade normal	125
5.3.2 Gerando valores com densidade de probabilidade normal.....	128
5.4 Exercícios para retenção de conhecimento.....	133
6. TRABALHANDO COM O EXCEL	137
6.1 Importar arquivos do Excel.....	137
6.2 Exportar arquivos para o Excel	139
6.3 Exercícios para retenção de conhecimento.....	140

7. PANDAS E A ESTRUTURA DOS DADOS	143
7.1 Como iniciar tratamento dos dados com a Pandas?	143
7.2 Configurando um DataFrame	143
7.3 Funções rápidas no DataFrame	148
7.4 Funções para cálculos estatísticos.....	151
7.5 Gráficos na Pandas	153
7.6 Os comandos de lógica para o DataFrame	159
7.6.1 Intersecção entre filtragens com “E”	161
7.6.2 Intersecção entre filtragens com “OU”	162
7.7 Estruturando colunas	163
7.7.1 Colocando uma coluna como índice	163
7.7.2 Reset o “index” mantendo o anterior como coluna	164
7.7.3 Reset o “index” apagando o anterior.....	165
7.7.4 Reset o “index” mantendo a coluna	165
7.7.5 Renomear uma coluna (“rename”).....	166
7.7.6 Apagando uma coluna (“drop”)	166
7.8 Operações com colunas do DataFrame.....	167
7.8.1 Função lambda para o DataFrame	169
7.8.2 Função lambda para diversas colunas	170
7.8.3 Função lambda para todo DataFrame com apenas uma regra	171
7.9 Localizar informações.....	172
7.9.1 O comando “loc”	173
7.9.2 O comando “iloc”	175
7.10 Exercícios para retenção de conhecimento.....	177
8. A FÍSICA E O PYTHON	185
8.1 Revisitando a física básica da cinemática	185
8.2 Quadratura numérica	192
8.3 Equações diferenciais	194

8.4	Funções (functions) no Python.....	197
8.5	Modelos com equações diferenciais no Python.....	198
8.6	Sistemas de equações diferenciais	200
8.7	Cosmologia quântica – Universos paralelos	205
8.8	Mecânica orbital – Viagem Terra-Marte.....	208
8.9	Rastreando asteroides com a Nasa	213
8.9.1	Preparando a base de dados dos asteroides	213
8.9.2	Estatística das descobertas de asteroide	216
8.9.3	Relação distância e diâmetro de asteroides	218
8.10	Exercícios para retenção de conhecimento.....	220
9.	CONTROLE DE FLUXO E ITERAÇÕES	225
9.1	Entrada e saída de dados	225
9.2	A utilização do if/else/elif.....	226
9.3	Controle de Iterações	231
9.3.1	Iteração com while.....	231
9.3.2	Iteração com for-range	234
9.4	Exercícios para retenção de conhecimento.....	237
	REFERÊNCIAS BIBLIOGRÁFICAS	241
	SOLUÇÕES DOS EXERCÍCIOS.....	243

CAPÍTULO 1

AMBIENTE DE PROGRAMAÇÃO

1.1 ESCOLHENDO O AMBIENTE DE PROGRAMAÇÃO

Ao iniciar o estudo de uma linguagem de programação, nós temos que escolher, nos dias atuais, um local para editar e rodar nossos programas. Nos anos primordiais da programação de computadores, essa escolha não existia, visto que a linguagem era única e a diferença entre as linhas de programação se dava na dependência do tipo de computador adquirido. O que se fazia era programar um código, por exemplo, em Fortran, usando um editor tipo bloco de notas, e depois, em separado, rodava-se um programa apenas para compilar e descobrir possíveis erros, e outro programa chamado de “linkador” para construir as bibliotecas necessárias para a execução correta.

Quando surgiram os primeiros microcomputadores, as duas primeiras empresas dominaram esse nicho esquecido pelas grandes corporações, e duas linhas diferentes para programação surgiram para os pequenos usuários: ou se programava em computadores com sistema da Apple, ou se programava em computadores com sistema da Microsoft.

Atualmente existem centenas de linguagens de programação, e para cada uma existem algumas dezenas de empresas que disponibilizam softwares específicos que facilitam a edição e a execução dos códigos.

Na linguagem de programação Python existem diversos ambientes, e, neles, embora a linguagem seja gratuita e “open source”, as empresas disputam o interesse dos programadores para formar um nicho ligado aos seus produtos. Assim, uma vez formada uma grande comunidade em torno do ambiente de programação, a empresa começa a oferecer produtos customizados para outras empresas que são dependentes da linguagem. Em suma, quanto maior o número de programadores ligados a um ambiente de programação, mais fácil e barata será a mão de obra para se contratar programadores e desenvolvedores de software.

Especificamente para o Python, temos três interessantes ambientes de programação, formados pelos colaboradores do Google (“Colab”), pelos produtos da Anaconda e, finalmente, pelo ambiente da Microsoft, conhecido como VSCode ou Visual Studio Code. Dentro desses ambientes ainda existem diferentes editores. Um editor bastante comum entre eles é conhecido como Jupyter, no qual a programação acontece no formato de células, cada qual podendo ser independente do restante da programação. No caso da Anaconda existem outros, como o Spyder, um editor de programação nos moldes dos antigos editores tipo “bloco de notas”, em que, após escrever o código, necessita-se rodar o programa para observar os resultados.

O importante da linguagem, no entanto, não é o ambiente, mas sim a forma lógica de raciocinar para se construir um bom programa, que execute de modo ótimo a tarefa proposta e que obtenha resultados rápidos.

1.2 AMBIENTE DE COLABORADORES DO GOOGLE (COLAB)

Esse é um ambiente que utiliza o editor estilo Jupyter para programar códigos em linguagem Python. Uma grande vantagem é não precisar instalar o programa, visto que está vinculado à sua conta no Google. Qualquer pessoa que possua uma conta no Google poderá entrar no ambiente do Colab e imediatamente iniciar o uso do editor para programar seu script.

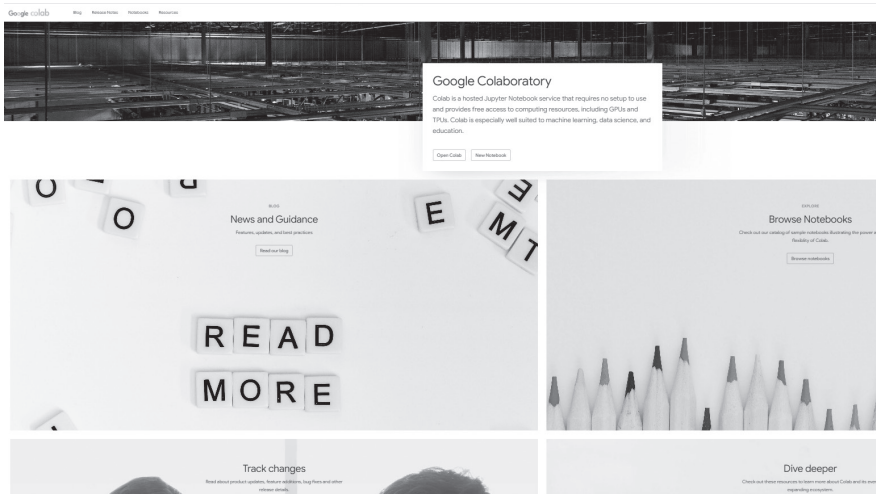


Figura 1.1 – Página principal do Google Colaboratory.

Fonte: <https://colab.google/>

Entrando com sua conta no Google Colab, a página principal (no período da elaboração deste livro) tinha o formato da Figura 1.2. Essa página mostra os conteúdos de seu Google Drive em que estão salvos os programas em Python e um botão que levará a um novo notebook. Um programa em ambiente Jupyter tem o nome de notebook, com extensão *.ipynb.

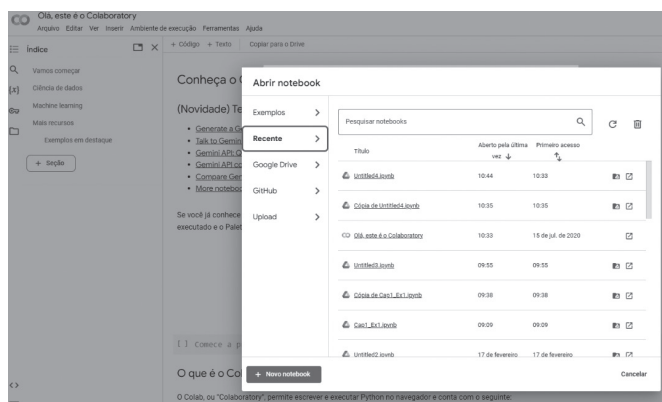


Figura 1.2 – Página inicial para um novo notebook.

Ao clicar em “Novo notebook” o usuário é levado ao ambiente da Figura 1.3, no qual a programação dos comandos deve ser inserida linha a linha, exatamente como apresentado na figura, em que se encontra a frase “Comece a programar ou gere código com IA”. Como esse é um sistema dinâmico que muda quase que diariamente, pode acontecer de, na data de publicação deste livro, a aparência da entrada ser um pouco diferente. Mas, mesmo com qualquer mudança, provavelmente a estrutura do Jupyter ainda será a mesma por um longo tempo.



Figura 1.3 – Início da programação em Jupyter no Colab.

As abas acima das linhas de programação, como “Arquivo”, “Editar”, “Ver”, “Inserir”, “Ambiente de execução”, “Ferramentas”, “Ajuda”, têm basicamente o mesmo formato tanto para o Colab como para a Anaconda e o VSCode.

1.3 AMBIENTE JUPYTER ANACONDA

Esse é um ambiente que precisa ser instalado diretamente da *site* da Anaconda e, dependendo do computador, poderá demorar entre 30 e 50 minutos. Essa é uma pequena desvantagem, no entanto, a Anaconda oferece não somente o Jupyter como também o editor Spyder e outras formas interessantes de utilizar a linguagem Python.

No momento em que este texto estava sendo elaborado, a página principal para baixar o ambiente de programação da Anaconda tinha a disposição da Figura 1.4. Isso muda de tempos em tempos, mas, existindo o botão “free download”, ao clicar nele, começa a transferência do arquivo para a pasta “downloads” do computador. Após isso, basta clicar no programa baixado que se iniciará a instalação. É nesse ponto que, dependendo do computador, o tempo para a completa instalação poderá demorar de 30 a 40 minutos (depende da memória Ram instalada no computador).

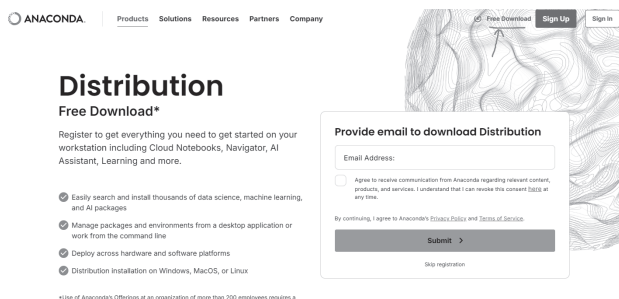


Figura 1.4 – Início da página de instalação da Anaconda.

Clicando no software instalado, o Anaconda Navigator terá a aparência da Figura 1.5. Entre as figuras da página principal, existem os *links* para se acessar o Jupyter e o Spyder, nos quais as linhas de programação dos códigos poderão ser executadas.

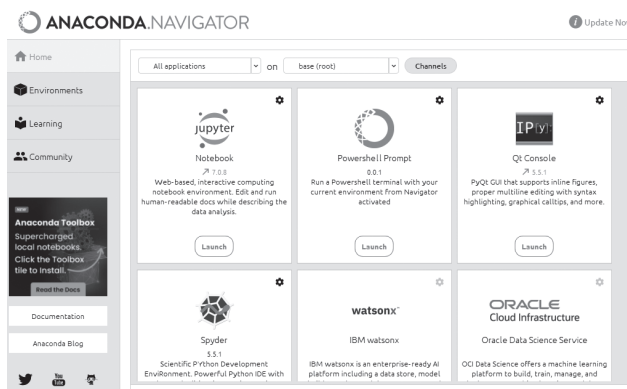


Figura 1.5 – Página inicial do Anaconda Navigator.

Programas no Jupyter, como no Colab, se chamam “notebook” e podem ser importados ou iniciados em branco dentro da aba “New”, como visto na Figura 1.6.

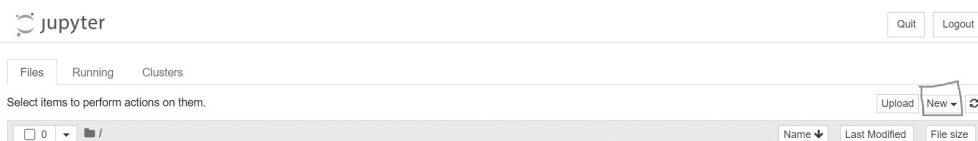


Figura 1.6 – Ambiente de programação em Jupyter (Anaconda).

Ao clicarmos em “New”, podemos criar um programa no formato do Jupyter, que terá como extensão *.ipynb, enquanto os programas em Python são salvos com extensão *.py. Na Figura 1.7, a tecla de início dá a possibilidade de se começar um programa em Python 3 (outras versões mais recentes terão outros nomes), em arquivo de texto (Text File), pastas novas para salvar programas (Folder), ou mesmo de se abrirem novos terminais de conexões para programações (Terminal).

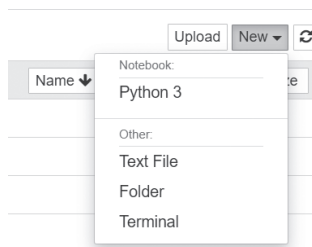


Figura 1.7 – Início de um novo programa.

Quando se clica em novo programa em Python, o nome dele aparece com “Untitled”, esperando para a programação na primeira linha, chamada de “In []”. As abas File, Edit, View etc. têm as mesmas funções do Python em programação no editor de texto. Na Figura 1.8, é possível observar a linha em branco esperando as primeiras anotações de um programa. Em versões mais recentes (por exemplo, IPython 8.25.0), a aba de abertura para novos programas aparece com o nome “notebook”, que deverá ser clicada para se abrir a mesma caixa apresentada na Figura 1.7.

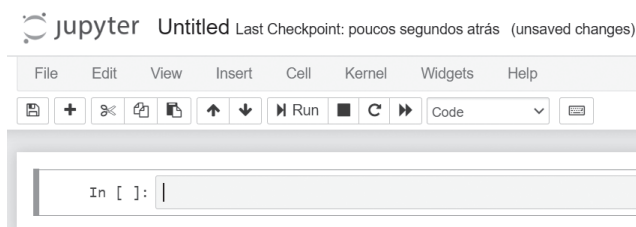


Figura 1.8 – Linhas e células à espera de um novo comando.

Para se nomear um programa no Jupyter ou renomear um já existente, basta clicar em cima da palavra “Untitled”, que se abrirá uma tela à espera de um nome para o programa que está sendo construído, como visto na Figura 1.9. Em cima do retângulo com o nome “Untitled”, coloca-se o nome desejado e clica-se no botão “Rename”.

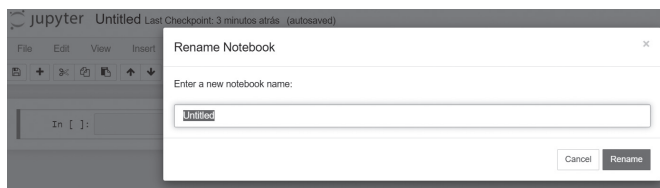


Figura 1.9 – Renomeando um programa no Jupyter.

1.4 PRIMEIROS PROGRAMAS EM JUPYTER

Começamos normalmente (mas isso não é uma obrigação) colocando as bibliotecas que serão importadas. Por exemplo, vamos supor que desejamos fazer um gráfico de um array no qual se tenha um vetor x e com ele se construa a função

$$f(x) = 2x^2 + x$$

Como apresentaremos nos capítulos seguintes deste livro, precisaremos importar diversas bibliotecas no Python, visto que elas não estão disponíveis como em outras linguagens de programação. O programador deverá, a cada novo programa, importar a biblioteca desejada para a execução do programa. Apenas como ilustração do funcionamento do Jupyter, para a simples equação anterior, precisamos da biblioteca NumPy e da biblioteca matplotlib.pyplot para o gráfico. Então, na primeira célula em branco do Jupyter, colocamos nossas bibliotecas, como visto na Figura 1.10.

Como passar para a próxima linha? No ambiente Jupyter, para que ele aceite a linha e salve na memória os passos da programação, precisaremos clicar em Shift+Enter a fim de que o programa comece a executar os comandos. Se o usuário apenas clicar em Enter, as células abrirão mais espaço em branco abaixo, mas dentro da própria célula, e não criando outra linha de programação.

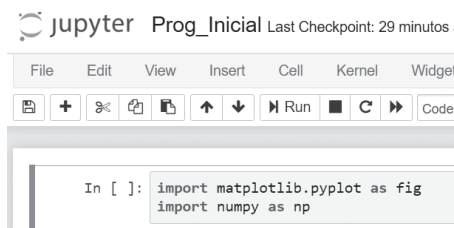


Figura 1.10 – Começando um programa na primeira célula.

Quando clicamos em Shift+Enter, o programa aceita a linha programada e abre uma nova linha. A antiga, então, começa a receber uma numeração, e, assim, a cada nova linha, as células vão recebendo seus números de linhas [1], [2], [3], ... etc., como visto na Figura 1.11.

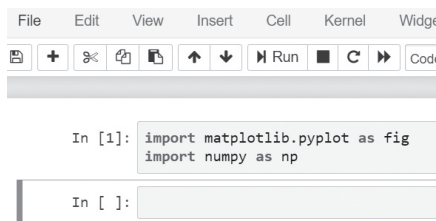


Figura 1.11 – Programa começa a executar os comandos de linha.

Primeiro, criamos o array para x , que será a base da função $f(x)$ da qual se deseja fazer um gráfico, como apresentado na Figura 1.12. Após isso, sabemos que podemos

programar a função $f(x)$. Ela não precisará ser em outra célula, mas pode ser visualmente mais interessante separar os valores das variáveis. É o programador quem decide o tipo de programa que desejará realizar, de forma a ficar mais didático e visualmente melhor.

```
In [1]: import matplotlib.pyplot as fig
import numpy as np
```

```
In [2]: x = np.arange(0, 5, 0.1)
```

Figura 1.12 – Programação da linha na qual se quer criar o eixo x para o gráfico de $f(x)$.

Após a geração do vetor x e do vetor y por meio da equação do segundo grau, precisaremos usar a biblioteca matplotlib.pyplot para a construção do gráfico com a função plot(), conforme será apresentado nos capítulos seguintes. Na Figura 1.13, é possível ver que o gráfico é plotado dentro da célula, usando a estrutura “inline”.

A cada alteração que for realizada na célula do gráfico ou em qualquer outra, a célula de cálculo precisará ser executada novamente com Shift+Enter. A Figura 1.13 apresenta o programa finalizado e completo, com as funções geradas e o gráfico plotado. Esse programa ainda pode ser reduzido colocando-se os vetores todos juntos na mesma célula. Como dito antes, tudo depende de como o programador quer exibir seu programa, ou mesmo reestudá-lo numa outra oportunidade.

```
In [1]: import matplotlib.pyplot as fig
import numpy as np

In [3]: x = np.arange(0, 5, 0.1)
x

Out[3]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])

In [4]: y = 2*x**2 + x
y

Out[4]: array([ 0. , 0.12, 0.28, 0.48, 0.72, 1. , 1.32, 1.68, 2.08,
2.52, 3. , 3.52, 4.08, 4.68, 5.32, 6. , 6.72, 7.48,
8.28, 9.12, 10. , 10.92, 11.88, 12.88, 13.92, 15. , 16.12,
17.28, 18.48, 19.72, 21. , 22.32, 23.68, 25.08, 26.52, 28. ,
29.52, 31.08, 32.68, 34.32, 36. , 37.72, 39.48, 41.28, 43.12,
45. , 46.92, 48.88, 50.88, 52.92])

In [5]: fig.plot(x,y,'-k')

Out[5]: [<matplotlib.lines.Line2D at 0x1dec9f03e48>]
```

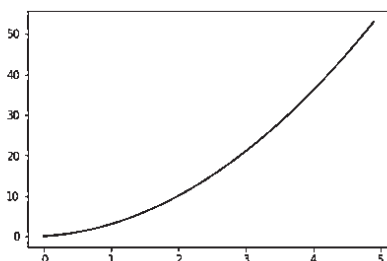


Figura 1.13 – Término do programa com os cálculos e gráfico.

1.5 ALTERANDO E EDITANDO CÉLULAS NO JUPYTER

Muitas vezes, depois de rodar o programa, percebemos que erramos uma determinada linha ou alguma fórmula dentro de uma célula inteira. O botão “Edit” do Jupyter permite muitas alterações, como a cópia de células, a exclusão, a procura de células com comandos específicos e sua troca por outras informações e o agrupamento de células. Para acessar a edição, basta clicar em “Edit”, como na Figura 1.14.

Isso é uma versão mais amigável do que os editores de texto do Python já fazem há muito tempo, mas são funções importantes e bem práticas de serem acessadas, mesmo por programadores iniciantes. Para se apagar uma célula, existem duas maneiras para a ação “delete”. A primeira é acessando o botão “Edit” e clicando em “Cut Cells”.

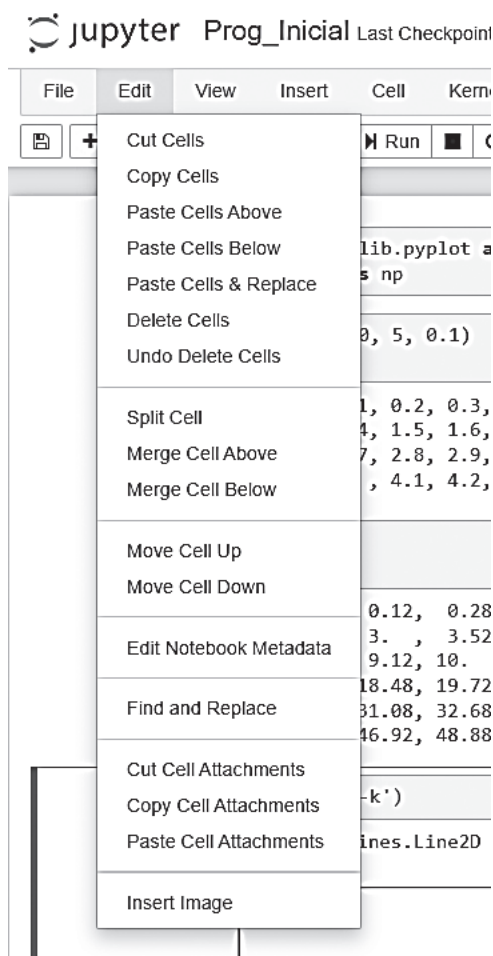


Figura 1.14 – Funções de edições no Jupyter.

A segunda maneira de se apagar uma célula inteira é selecionando-a, como na Figura 1.15, e clicando no desenho de uma pequena tesoura, na parte superior dos comandos do Jupyter. Isso apagará todas as informações e desaparecerá com a programação.

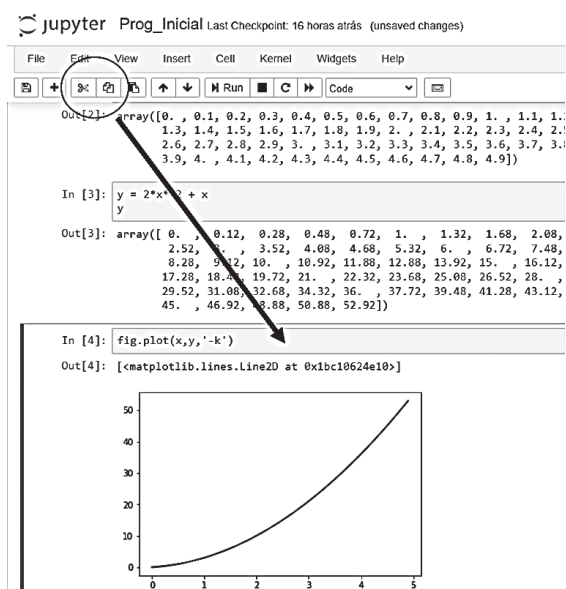


Figura 1.15 – Apagando uma célula.

Caso se tenha apagado erroneamente uma célula inteira de informação com programação, basta clicar novamente no botão “Edit” e, em seguida, escolher “Undo Delete Cells”, como apresentado na Figura 1.16. Em versões mais recentes, apenas clicando em “Undo”, ou então no comando de teclado “CTRL + Z”, é possível desfazer a ação anterior. No mesmo botão, para se copiar uma célula, basta selecionar com o mouse e, em seguida, escolher o tipo de cópia desejado para a célula.

Pode-se copiar uma célula com programação colocando-se uma linha acima do local que se está (“Paste Cells Above”); pode-se colocar abaixo do local em que o mouse permanece selecionado (“Paste Cells Below”); ou, ainda, pode-se passar a célula do programa no lugar em que já existe uma outra célula, escolhendo “Paste Cells & Replace”.

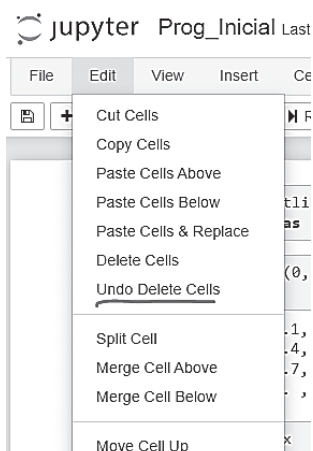


Figura 1.16 – Recuperando células que foram excluídas por engano.

Por exemplo, se copiarmos a célula [4] de nosso exemplo e selecionarmos para colocar na célula abaixo, o resultado serão dois gráficos, pois são copiados não apenas os gráficos, mas também a programação que os gerou, como na Figura 1.17.

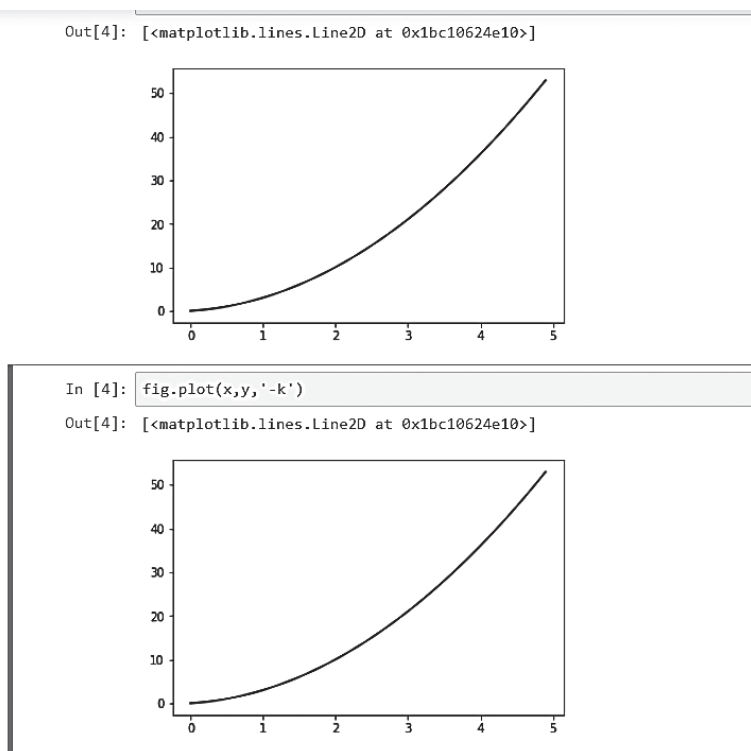


Figura 1.17 – Copiar e colar uma célula na linha abaixo.

Como no Microsoft Word, podemos, ainda, usar o botão de “Copiar”, tradicionalmente conhecido, para copiar uma célula, como na Figura 1.18.

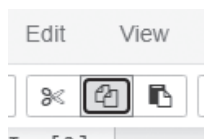


Figura 1.18 – Botão de copiar células.

Para colar, basta usar o botão também existente no Word, como na Figura 1.19.



Figura 1.19 – Botão de colar células no Jupyter.

Logo, de maneira geral, basta sempre escolher com o mouse uma determinada célula e clicar num dos botões para se ter o resultado desejado, como apresentado na Figura 1.20.

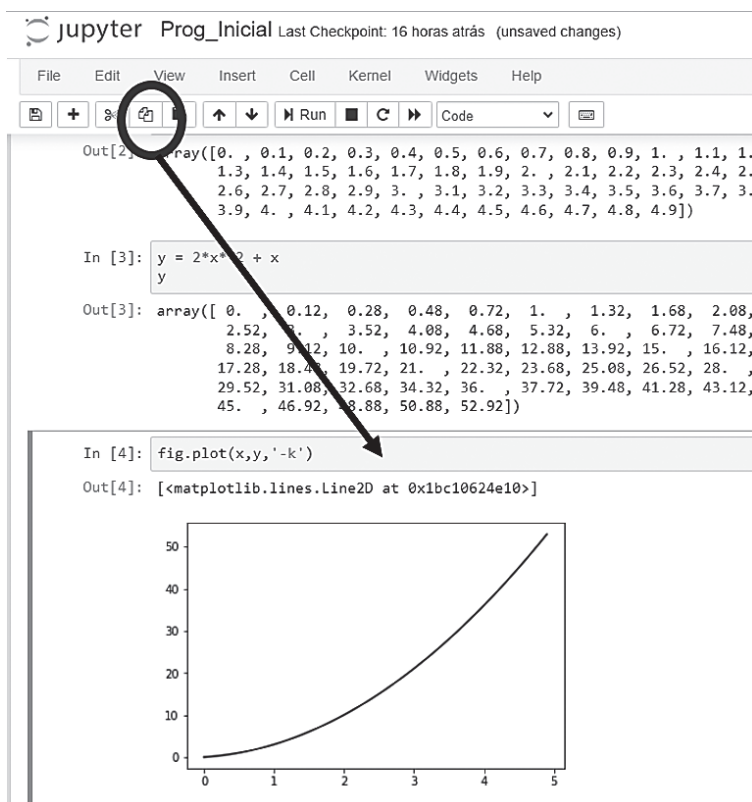


Figura 1.20 – Copiar células com programação usando o botão de copiar no Jupyter.

1.6 UTILIZAÇÃO DO KERNEL

Alterações feitas em células podem ser executadas novamente usando Shift+Enter, como dito antes. Mas, às vezes, temos muitas alterações feitas em células com informações que estão abaixo do local em que se alterou, ou logo no início do programa.

Nesses casos, por segurança, é melhor executar o programa todo, de uma só vez, em vez de se selecionar célula por célula com as programações. Para isso, devemos acessar o botão chamado de *Kernel*. O Kernel, no Python ou no Jupyter, é uma máquina interna para armazenar informações no *buffer* e executar programas seguindo a lógica do compilador. É ele que pode acelerar um programa ou retardar, caso os cálculos sejam muito complexos e demorados.

O Kernel é acessado como apresentado na Figura 1.21. Com ele, por exemplo, quando programas “travam”, podemos escolher “Interrupt”, ou mesmo “Restart”, para limparmos a memória de erros passados e começarmos a execução novamente. Em versões mais recentes, esse comando foi renomeado como “Restart Kernel” dentro da aba “Kernel”.

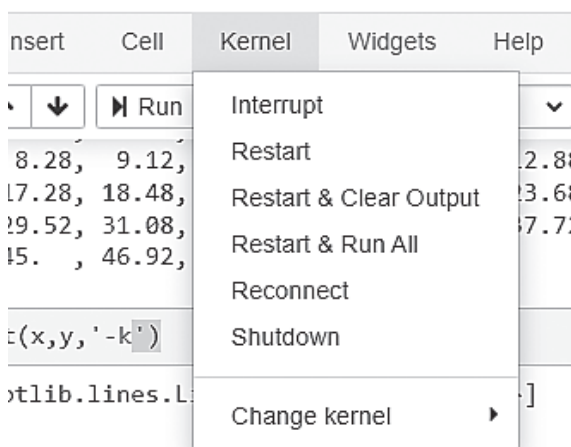


Figura 1.21 – Funções do Kernel.

Outra função importante do Kernel é limpar todas as células e reiniciar a execução automática “Restart & Clear Output”, ou “Restart & Run All”. Por exemplo, quando clicamos em nosso programa para “Restart & Clear Output”, o Jupyter limpa todas as células, deixando a numeração das linhas em colchetes com branco interno [], indicando que o que ficou é apenas linha de programação que não foi executada, como apresentado na Figura 1.22.

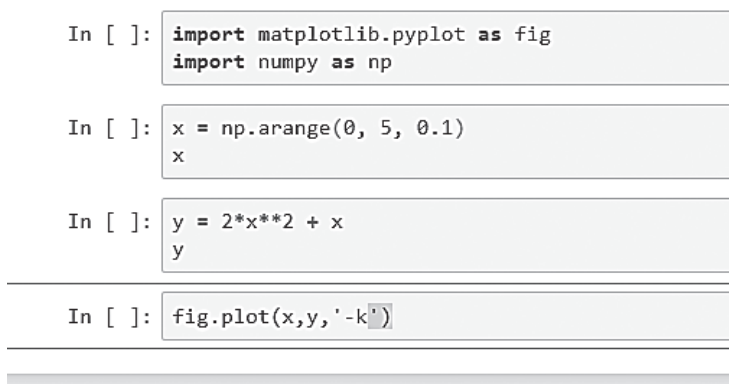


Figura 1.22 – Função “Restart & Clear Output” no programa-exemplo.

Para rodar todas as células sem apagar a memória e sem usar o Kernel para limpá-las, basta acessar a aba “Cell” e escolher “Run All”. O Jupyter executará todas as linhas, mantendo na memória todos os cálculos existentes. Ainda se pode rodar partes do programa de forma automática, executando-se apenas linhas acima, com “Run All Above”; ou executando-se apenas linhas abaixo do local em que se está, com “Run All Below”.

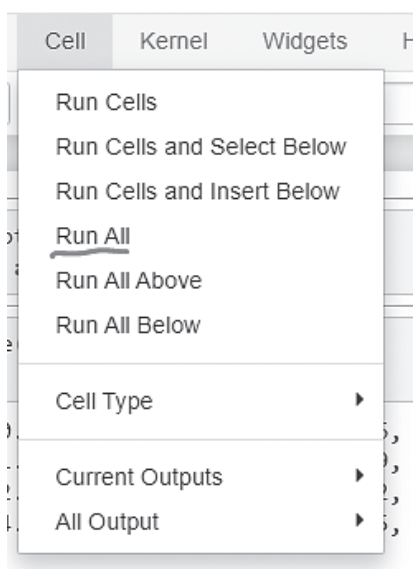


Figura 1.23 – Aba das funções “Cell” para execução das linhas.

Como pode ser visto na Figura 1.23, ainda podemos rodar parcialmente algumas linhas escolhidas pelo mouse clicando em “Run Cells and Select Below”. Na versão mais recente, quando este livro estava sendo escrito (IPython 8.25.0), a aba “Cell” foi substituída pela aba “Run”, mas se mantiveram todas as funções das versões anteriores.

1.7 SALVAR E DOWNLOAD

O Jupyter ainda fornece alternativas para salvamentos dos programas em diferentes formas. Esse tipo de salvamento permitirá recuperar e rodar os programas em outras ocasiões, usando-se ou não o ambiente do Jupyter.

O botão para abrir programas já salvos, salvar, copiar, renomear, entre tantas outras operações, é o “File”, como apresentado na Figura 1.24. A escolha desse botão sempre se dá quando iniciamos um novo programa ou quando terminamos a programação e desejamos salvar em alguma pasta particular local ou em algum terminal ligado ao computador em que estamos trabalhando.

Para novos programas, também acessamos o “File”, mas cabe salientar que, no Jupyter, os programas são chamados de “Notebooks”. Quando escolhemos novos programas, ou “New Notebook”, aparecerá a opção da linguagem Python. No caso deste texto, nossa versão é o Python 3, como pode ser conferido na Figura 1.25. Versões mais recentes ou atualizadas do Python aparecerão nessa tela com a numeração ou a versão do Python instalada no computador.

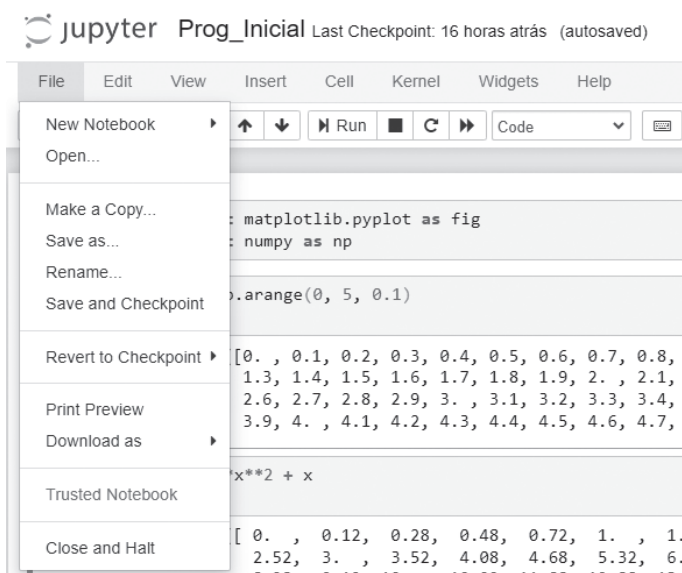


Figura 1.24 – Botão “File” para diversos tipos de salvamentos dos programas.

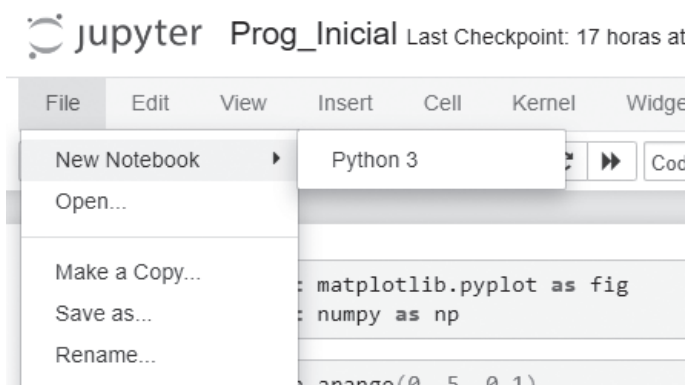


Figura-1.25 – Opção “New” para iniciar novo notebook (novo programa).

Na versão mais recente, enquanto este texto estava sendo escrito (IPython 8.25.0), a aba “File” passou a conter uma nova aba chamada “New”, na qual se pode abrir notebooks já feitos, o terminal, o console, arquivos e assim por diante. Na abertura de um novo notebook (um novo programa no ambiente Jupyter), a versão recente agora tem uma aba chamada “New Console for Notebook” dentro da aba “File”. Podemos salvar os programas (chamados de notebook) usando “Save as” quando desejarmos trocar os nomes dos programas em alguma pasta diferente da que estamos usando, ou mesmo após modificações importantes no programa ainda aberto. Na versão mais recente, a aba mudou para “Save Notebook As”.

Outra forma diferente de salvar nossos projetos ou programas é por meio da opção “Download”. Esse tipo de salvamento serve para que nosso programa seja aberto em outros ambientes, ou até em outras linguagens de programação.

O natural é salvar como Jupyter, usando “Notebook(.ipynb)”, mas podemos salvar para rodar na versão editor de texto do tipo Spyder, usando apenas a saída como formato Python, por meio de “Python(.py)”, como visto na Figura 1.26. Nesse caso, o programa será salvo na pasta “Downloads” do Windows e, de lá, poderá ser transferido para o local que o programador desejar.

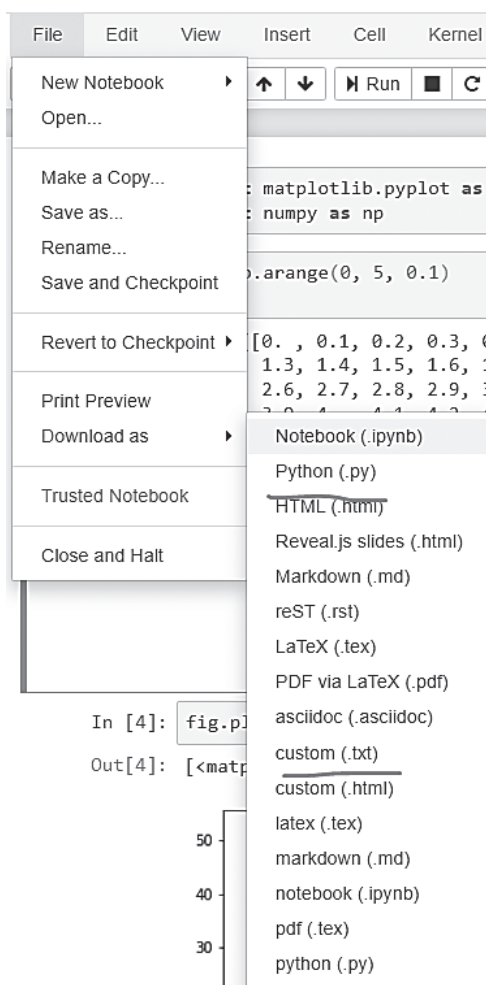


Figura 1.26 – Opção “Download” para salvar programas em diversos formatos.

Como visto, o ambiente de programação em Jupyter é muito interessante e amigável. Ele facilita a compreensão do funcionamento de linhas de código em Python até mesmo para não programadores. No entanto, ainda assim, o importante não é o ambiente de programação do Python, mas o conhecimento lógico de todos os passos para se resolver um problema com programação de script.

O conhecimento acadêmico e crítico independe do ambiente ao qual estamos expostos para programar; é necessário ainda um longo e duro preparo na elaboração de programas, para se evitarem os erros que serão discutidos ao longo dos capítulos.

O que você viu neste capítulo:

- Usar o ambiente Jupyter
- Editar um Notebook no Jupyter
- Copiar células no Jupyter
- Salvar documentos
- Rodar programas pequenos
- Reset do Kernel
- Limpar células
- Abrir novos documentos

IMERSÃO EM PYTHON

PYTHON ■ COMPUTAÇÃO ■ PROGRAMAÇÃO ■ SIMULAÇÃO COMPUTACIONAL ■
LINGUAGEM DE PROGRAMAÇÃO

Esta obra introduz o leitor ao estudo da computação usando o Python no ambiente de programação Jupyter. O texto parte da premissa de que o leitor não sabe programar e deseja aprender de forma contínua, no seu ritmo de aquisição de conhecimento. Os códigos e bibliotecas utilizadas são simples, com exemplos e exercícios diretos e rápidos, para permitir que o leitor se concentre melhor na formulação e execução dos scripts em Python.

Além das bibliotecas e assuntos básicos abordados, *Imersão em Python* oferece ainda descrições sobre o uso avançado da programação em assuntos que vão desde a matemática, física, administração e economia até os conceitos sobre banco de dados e estatística.

O que você vai encontrar:

- Como trabalhar com Python no ambiente Jupyter.
- Manipulação de dados em listas.
- Conceito e aplicações diretas de array.
- Gráficos básicos, estatísticos e avançados.
- Bibliotecas de matemática, estatística, Numpy, Pandas e Matplotlib.
- Importação e exportação de dados do Python para Excel.
- Rastreamento de asteroides com biblioteca da Nasa.
- Introdução a controle de fluxo, lógica de decisão e iterações.

O autor usa como poucos sua habilidade narrativa e pedagógica para proporcionar uma experiência de aprendizado original, sem perder clareza e objetividade, muito adequada para aqueles que têm interesse em aprender de forma rápida e efetiva.

Prof. Dr. Carlos Henrique C. Ribeiro

Divisão de Ciência da Computação
Instituto Tecnológico de Aeronáutica (ITA)

MARCO ANTONIO LEONEL CAETANO

Matemático pela Universidade Estadual Paulista (Unesp) e mestre e doutor em Engenharia Aeronáutica pelo Instituto Tecnológico de Aeronáutica (ITA). Foi professor convidado de pós-graduação do ITA de 2004 a 2024. Foi avaliador de curso superior do Ministério da Educação até 2010, diretor-tesoureiro da Sociedade Brasileira de Automática (SBA) e coordenador de projetos e assessor da Fundação de Amparo à Pesquisa do Estado de São Paulo (Fapesp). Em 2023 recebeu do Insper o prêmio Chafi Haddad de Excelência em Ensino em Economia. Autor de outros livros também pela Blucher, sendo *Éden dos algoritmos em Python* semifinalista do prêmio Jabuti de 2025.

Saiba mais sobre o livro



Blucher

www.blucher.com.br

