Outros livros da série *Use a Cabeça* [alguns ainda sem publicação no Brasil]

Use a Cabeça Desenvolvimento para Android

Use a Cabeça C#

Use a Cabeça Análise de Dados

Use a Cabeça Padrões de Projetos

Head First Git

Head First Go

Use a Cabeça Java

Use a Cabeça Programação JavaScript

Head First Kotlin

Use a Cabeça Aprenda a Programar

Use a Cabeça Análise e Projeto Orientado ao Objeto

Use a Cabeça Programação

Use a Cabeça Desenvolvimento de Software

Use a Cabeça SQL

Use a Cabeça Estatística

Head First Swift

Use a Cabeça Web Design

Elogios à Terceira Edição do *Use a Cabeça Python*

"O Python existe desde 1991, mas ultimamente voltou com força total devido ao aprendizado de máquina e aos ambientes Jupyter. Como essa linguagem mudou ao longo de muitos anos, pode ser difícil saber como começar. O *Use a Cabeça Python* não mostra só o básico, mas corta toda a parte desnecessária. Você gostará de desenvolver um aplicativo real com uma história divertida e real enquanto cria notebooks e implanta um aplicativo viável na web. Se o Python está em sua lista de tarefas, comece com este livro!"

— **Daniel Hinojosa, desenvolvedor/instrutor/apresentador**

"Um ótimo começo para a poderosa linguagem de programação do Python, levando você em uma jornada educativa e intrigante ao criar um aplicativo desde o conceito até um webapp online."

— **Michael Hopkins, geocientista profissional, PMP**

"O *Use a Cabeça Python* proporciona uma experiência de aprendizagem envolvente, parecendo que um tutor amigo e experiente o guia pessoalmente. Com um equilíbrio perfeito de um conteúdo divertido e informativo, este livro torna o aprendizado da linguagem Python divertido e eficaz."

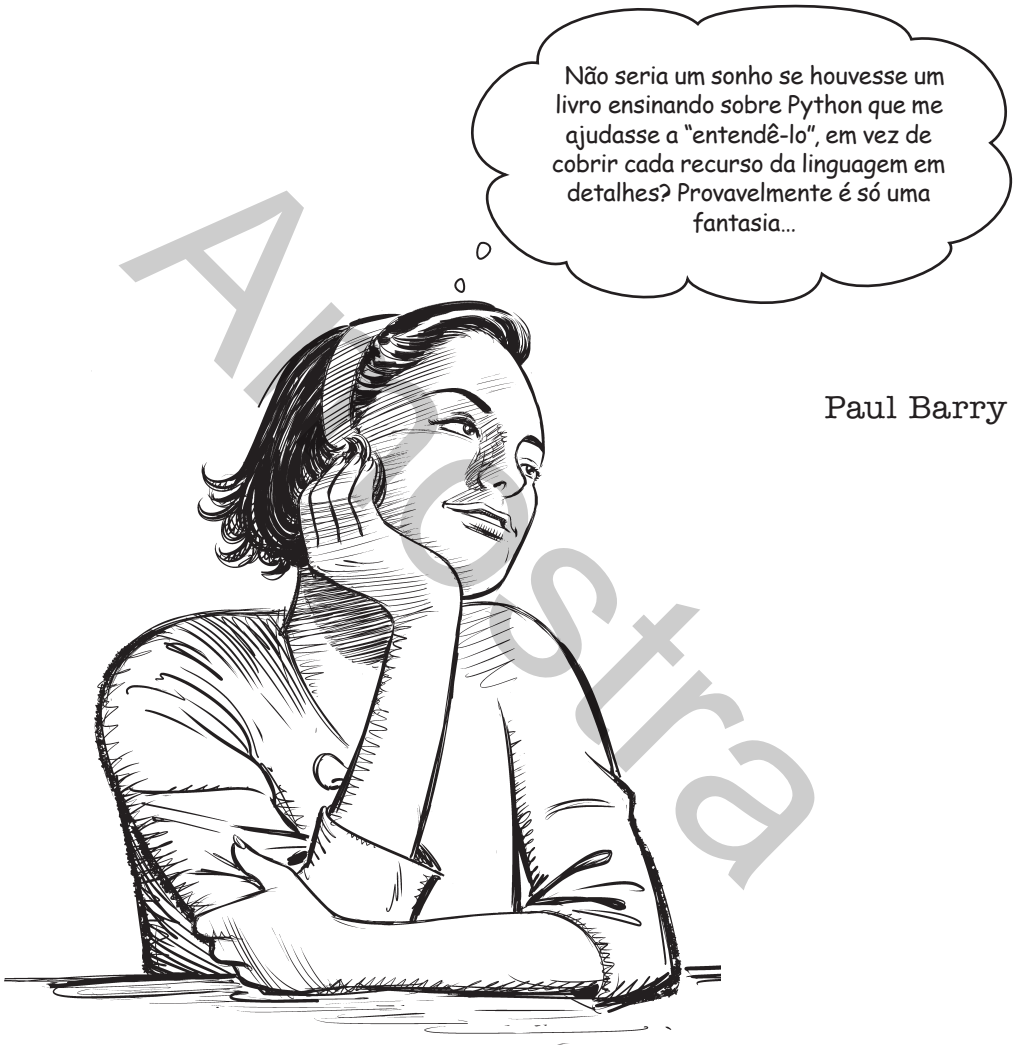
— **William Jamir Silva, Anaconda, Inc., engenheiro de software**

"Uma forma divertida de aprender a programar com Python, usando as mesmas ferramentas de desenvolvimento que meus colegas utilizam no dia a dia. O livro inclui uma progressão de desafios que me levaram a resolver problemas que eu não conseguia abordar no início. Ele tem uma introdução sucinta e valiosa para a codificação 'no estilo Python' que reconhece que há 'mais de uma forma de fazê-lo'."

— **Dave Marsden, arquiteto de nuvem, CTS**

Use a Cabeça Python™

Terceira Edição



Paul Barry

ALTA BOOKS
EDITORA
Rio de Janeiro, 2025

Use a Cabeça Python – 3ª Edição

ISBN: 978-85-508-2687-5

Impresso no Brasil — 3ª Edição, 2025 — Edição revisada conforme o Acordo Ortográfico da Língua Portuguesa de 2009.

Dados Internacionais de Catalogação na Publicação (CIP)

B281p
3.ed. Barry, Paul
Use a Cabeça Python: o guia de aprendizagem para os fundamentos da programação em Python/ Paul Barry ; tradução de Eveline Machado. – 3.ed. – Rio de Janeiro: Alta Books, 2025.
656 p.; il.; 17 x 24 cm.

Título original: *Head First Python*.
ISBN 978-85-508-2687-5

1. Python (Linguagem de programação). 2. Programação de computadores. 3. Desenvolvimento de software. 4. Ciência de dados. I. Machado, Eveline. II. Título.
CDD 005.133

Índice para catálogo sistemático:

1. Python (Linguagem de programação) 005.133
2. Programação de computadores 005.1

Grupo Editorial Alta Books

Diretor Editorial: Anderson Vieira.
Vendas ao Governo: Cristiane Mutüs.
Gerência Marketing: Viviane Paiva.

Produtora Editorial: Isabella Gibara.
Tradução e Copidesque: Eveline Machado.
Revisão Gramatical: Denise Himpel.
Diagramação: Ana Lúcia Quaresma.
Revisão Técnica: Jhonatan Pereira.
(Desenvolvedor de Software)

ALTA BOOKS
GRUPO EDITORIAL

Rua Viúva Cláudio, 291 — Bairro Industrial do Jacaré
CEP: 20.970-031 — Rio de Janeiro (RJ)
Tels.: (21) 3278-8069 / 3278-8419
www.altabooks.com.br — altabooks@altabooks.com.br
Ouvidoria: ouvidoria@altabooks.com.br

Editora **afiliada à:**

Dedicado, mais uma vez, à comunidade Python.

A todas as pessoas generosas que continuam trabalhando para fazer do Python a maravilhosa tecnologia de programação que ele é. Obrigado.

Paul Barry

Paul Barry vive com sua esposa Deirdre em Carlow, Irlanda, uma pequena cidade localizada a 80km de Dublin. Seus três filhos adultos (Joseph, Aaron e Aideen) "deixaram o ninho" recentemente.

Paul trabalha na South East Technological University (SETU), localizada no Kilkenny Road Campus, Carlow, onde dá palestras fazendo parte do Departamento de Computação acadêmico. Paul deu aulas por *muito* tempo, e vem usando a linguagem Python com todos os seus alunos por quase quinze anos.

Paul tem mestrado e licenciatura em Computação, e é pós-graduado em Aprendizagem e Ensino. Ele nunca chegou a fazer doutorado, por isso ninguém deve se referir a ele como "professor", mas ele se diverte quando acontece.

Paul passou parte dos anos 1980 e 1990 trabalhando no setor de TI, principalmente na área de saúde no Canadá. Ele também escreveu outros livros e — na época — foi editor colaborador na revista *Linux Journal*.

Tudo isso (infelizmente) significa que Paul *está envelhecendo um pouco*. Por favor, não conte a ninguém.

Índice Remissivo (Sumário)

Conteúdo

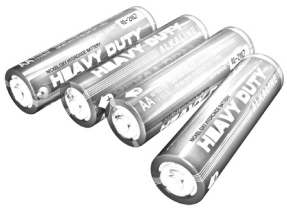
Introdução

Seu cérebro com Python. Aqui você está tentando aprender algo e aqui seu cérebro está lhe fazendo um favor assegurando que a aprendizagem não se fixe. Seu cérebro está pensando: "É melhor deixar espaço para coisas mais importantes, como quais animais selvagens evitar e se fazer snowboard nu é uma má ideia." Então, como fazer seu cérebro pensar que sua vida depende de saber programar em Python?

por que Python?

0 Parecido, mas Diferente

O Python começa contando do zero, o que deve ser familiar. Na verdade, o Python tem muito em **comum** com outras linguagens de programação. Existem **variáveis**, **loops**, **condicionais**, **funções** e afins. Neste capítulo de abertura, faremos um **tour avançado e rápido** pelos conceitos básicos do Python, apresentando a linguagem sem detalhar muito. Você aprenderá a **criar** e **executar** código com o Jupyter Notebook (rodando no VS Code). Verá como muitas funcionalidades de programação vêm **predefinidas** no Python, que você **utilizará** para fazer as coisas. Também aprenderá que, embora o Python compartilhe muitas de suas ideias com outras linguagens de programação, como elas se manifestam em seu código pode ser bem **diferente**. Agora, não entenda mal: estamos falando sobre algo **bom** e diferente, não sobre uma diferença *ruim*. Leia para saber mais…



mergulhando

1 Cause um Impacto

A melhor forma de aprender uma nova linguagem é programando. E se você for escrever código, precisará de um problema **real**. Por sorte, temos um. Neste capítulo, você começará sua jornada de desenvolvimento de aplicativos Python causando impacto em nosso grande amigo, **Coach de Natação**. Você começará com as **strings** Python, aprendendo a **manipulá-las** como quiser, tentando produzir uma solução baseada em Python para o problema do Coach. Também verá mais da estrutura de dados **lista** predefinida do Python, aprenderá como as **variáveis** funcionam e descobrirá como ler as **mensagens de erro** do Python sem surtar, ao mesmo tempo resolvendo um problema *real* com código Python *real*. Mergulharemos (de cabeça)…

listas de números

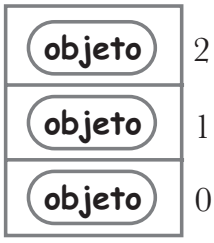
2 Processando Dados de Listas

Quanto mais código você escreve, melhor fica. Simples assim. Neste capítulo, você continua a criar código Python para ajudar o Coach. Você aprende a **ler** os dados de um **arquivo** de dados fornecido pelo Coach, colocando suas linhas em uma **lista**, uma das **estruturas de dados** predefinidas mais poderosas do Python. Além de criar listas a partir dos dados do arquivo, também aprenderá como criar listas do zero, **aumentando** a lista **dinamicamente** conforme a necessidade. E processará as listas usando uma das construções de loop mais populares do Python: o loop **for**. Você **converterá** os valores de um formato de dados em outro e fará um novo melhor amigo (o próprio **BFF** Python). Já chega de conversa fiada; é hora de arregaçar as mangas e voltar ao trabalho.

lista de arquivos

3 Funções, Módulos e Arquivos

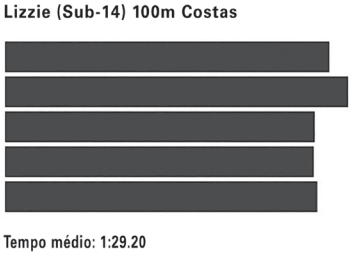
O código não pode viver em um notebook para sempre. Ele quer ser livre. E quando se trata de libertar seu código e **compartilhá-lo** com outras pessoas, uma **função** personalizada é o primeiro passo, seguida logo atrás por um **módulo**, que permite organizar e compartilhar seu código. Neste capítulo, você criará uma função diretamente a partir do código que escreveu até agora e, no processo, também criará um módulo **compartilhável**. Você colocará seu módulo imediatamente em operação quando processar os dados de natação do Coach com os loops **for**, as declarações **if**, os testes condicionais e a **PSL** (*Biblioteca Padrão do Python*). Aprenderá a **comentar** suas funções também (sempre uma *boa ideia*). Há muito o que fazer, então vamos lá!



literais de string formatadas

4 Crie Gráficos com Dados

Às vezes são as abordagens mais simples que fazem o trabalho. Neste capítulo você finalmente começa a produzir gráficos de barras para o Coach. Você fará isto usando apenas **strings**. Você já sabe que as strings do Python têm uma **bondade predefinida** e as **literais de string formatadas** do Python, ou seja, as *f-strings*, melhoram o que é possível de formas bem organizadas. Pode parecer estranha a proposta de criar gráficos de barra com texto, mas, como está prestes a descobrir, não é tão *absurdo* quanto parece. Ao longo do caminho você usará o Python para criar **arquivos**, bem como inicializar um navegador com apenas algumas linhas de código. Por fim, o Coach consegue o que quer: a **geração automática** de gráficos a partir de seus dados dos tempos de natação. Vamos lá!



organizando

5 Escolhendo uma Estrutura de Dados

Seu código precisa colocar os dados em algum lugar na memória. E quando se trata de organizar os dados em algum lugar… **na memória**, sua escolha de qual estrutura de dados usar pode ser crítica e muitas vezes é a diferença entre uma solução bagunçada que *funciona* e uma solução **elegante** que *funciona bem*. Neste capítulo, você aprenderá outra estrutura de dados predefinida do Python, o **dicionário**, que costuma ser combinado com a lista onipresente para criar estruturas de dados **complexas**. O Coach precisa de uma maneira fácil de selecionar os dados do nadador, e quando você coloca os dados do Coach em um dicionário, as pesquisas são moleza!

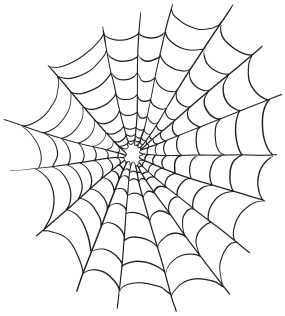
Dicionário

criando um webapp

6 Desenvolvimento para Web

Pergunte a dez programadores qual framework para web usar… e provavelmente terá onze respostas conflitantes! 😉

Quando se trata de desenvolvimento para web, o Python tem muitas *escolhas* tecnológicas, cada uma com uma comunidade de desenvolvedores leal e solidária. Neste capítulo, você verá o **desenvolvimento para web**, criando rapidamente um webapp para o Coach que exibe os dados do gráfico de barras de qualquer nadador. Ao longo do caminho, aprenderá a usar **templates** HTML, **decoradores** de funções, métodos HTTP **get** e **set** etc. Não há tempo a perder: o Coach está ansioso para mostrar seu novo sistema. Vamos lá!



implantação

7 Rode seu Código em Qualquer Lugar

Fazer com que o código seja executado no seu computador é uma coisa… O que você realmente quer é **implantar** seu código para que seja fácil para os usuários executá-lo também. E se consegue fazer isso *sem* muita complicação, bem melhor. Neste capítulo, você termina o webapp do Coach adicionando um pouco de **estilo**, antes de **implantá-lo** na **nuvem**. Mas, não vá pensando que é algo que aumenta a complexidade para 11 — longe disso. Uma edição anterior deste livro se gabou de que a implantação levava "cerca de 10 minutos", e ainda é um trabalho rápido agora… embora neste capítulo você implante seu webapp em 10 **etapas**. A nuvem está esperando para hospedar o webapp do Coach. Vamos lá!

trabalhando com HTML

8 Extraindo da Web

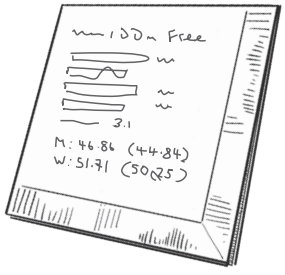
Em um mundo perfeito, seria fácil acessar todos os dados de que você precisa. Mas raramente é assim. Caso em questão: os dados são publicados na web. Os dados incorporados em HTML são projetados para serem **renderizados** por navegadores e **lidos** por pessoas. Mas e se você precisar **processar** os dados incorporados em HTML com o código? Azar seu? Bem, por sorte, o Python brilha quando se trata de **extrair** dados de páginas da web, e neste capítulo você aprenderá a fazer exatamente isso. Também aprenderá a **analisar** essas páginas HTML extraídas para **obter** dados úteis. Ao longo do caminho, verá o **fatiamento de listas** e a **biblioteca Beautiful Soup**. Mas, não se preocupe, isso ainda é o *Use a Cabeça Python*, não o *Use a Cabeça Cozinhando*...

trabalhando com dados

9

Manipulação de Dados

Às vezes seus dados não estão organizados como deveriam. Talvez você tenha uma *lista de listas*, mas realmente precisa de um *dicionário de dicionários*. Ou talvez precise relacionar um valor em uma estrutura de dados a um valor em outra, mas os valores não correspondem. Afff, é tão frustrante. Não se preocupe: o poder do Python está aqui para ajudar. Neste capítulo, você usará o Python para **manipular** seus dados e transformar os dados extraídos no final do capítulo anterior em algo realmente *útil*. Verá como **explorar** o dicionário do Python como uma tabela de **pesquisa**. Há conversões, integrações, atualizações, implantações e muito mais neste capítulo. E, no final, o esboço do Coach se torna *realidade*. Não pode esperar? Nem nós… vamos lá!



trabalhando com ~~elefantes~~ dataframes

9 1/2 Dados Tabulares

Às vezes é como se todos os dados do mundo quisessem estar em uma tabela. Os dados tabulares estão *em toda parte*. Os recordes mundiais de natação do capítulo anterior são dados **tabulares**. Se você tem idade suficiente para se lembrar das listas telefônicas, os dados são tabulares. Extratos bancários, faturas, planilhas, adivinha? Todos são dados tabulares. Neste *breve* capítulo, você aprenderá um pouco sobre uma das bibliotecas de análise de dados tabulares mais populares em Python: **pandas**. Você só tocará na superfície do que o pandas pode fazer, mas aprenderá o suficiente para conseguir explorar a estrutura de dados mais usada do pandas, o **dataframe**, quando precisar processar um bloco de dados tabulares.

bancos de dados

10 Organizando

Cedo ou tarde, os dados do seu aplicativo precisam ser gerenciados. E quando você precisa **gerenciar** seus dados de forma mais adequada, o Python (sozinho) pode não ser suficiente. Quando isso acontecer, você precisará acessar seu mecanismo de **banco de dados**. Para manter as coisas… em, eh… *gerenciáveis*, ficaremos com os mecanismos de banco de dados que suportam o bom e velho **SQL**. Neste capítulo, você não apenas **criará** um banco de dados e adicionará algumas **tabelas** a ele, mas também irá **inserir**, **selecionar** e **excluir** dados do seu banco de dados, executando todas essas tarefas com consultas SQL orquestradas por seu código Python.

list comprehensions

11 Integrações do Banco de Dados

Com as tabelas do banco de dados prontas, é hora de integrar. Seu webapp pode ter a **flexibilidade** que o Coach requer usando conjuntos de dados em suas tabelas do banco de dados, e neste capítulo você cria um módulo de **utilitários** que permite ao webapp **explorar** seu mecanismo do banco de dados. E, em uma busca interminável para fazer mais com menos código, você aprenderá a ler e gravar **list comprehensions** (compreensões de lista) que são o verdadeiro superpoder do Python. Também **reutilizará** muito do seu código preexistente de maneiras novas e interessantes, então vamos começar. Há muito trabalho de **integração** a fazer.


```
[sql for sql in dir(queries) if not sql.startswith("__")]
```

implantação revisitada

12 Toques Finais...

Você está chegando ao fim da sua jornada em Python. No capítulo final deste livro, você ajusta seu webapp para usar **MariaDB** como seu banco de dados de back-end, não o SQLite, então ajusta as coisas para implantar a versão mais recente do seu webapp no PythonAnywhere. Ao fazer isso, o Coach obtém acesso a um sistema que suporta **quaisquer** nadadores participando de **quaisquer** sessões de natação. Não há muita novidade sobre o Python no capítulo, já que você passa a maior parte do seu tempo ajustando o código existente para trabalhar com MariaDB e PythonAnywhere. O seu código Python nunca existe **isoladamente**: ele **interage** com o ambiente e os sistemas dos quais depende.


```
mysql> select count(*) from events;
+----------+
| count(*) |
+----------+
|       14 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from swimmers;
+----------+
| count(*) |
+----------+
|       23 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from times;
+----------+
| count(*) |
+----------+
|      467 |
+----------+
1 row in set (0.00 sec)
```

Apêndice

Dez Coisas Principais que Não Cobrimos

Acreditamos muito que é importante saber quando parar.

Especialmente quando o autor é da *Irlanda*, uma nação famosa por produzir indivíduos com o dom de não saber bem quando parar de falar. 😉 Adoramos falar sobre Python, nossa linguagem de programação favorita. Neste apêndice, apresentamos as dez principais coisas que, dadas cerca de seiscentas páginas ou mais, gostaríamos de detalhar no *Use a Cabeça* para lhe contar tudo. Há informações sobre classes, tratamento de exceções, testes, uma morsa (sério, uma *morsa*? Sim: uma *morsa*), switches, decoradores, gerenciadores de contexto, simultaneidade, type hints, ambientes virtuais e ferramentas do programador. Como dissemos, há sempre mais para falar. Então, vá em frente, vire a página e aproveite as próximas páginas!

Introdução

Nesta seção, responderemos à pergunta que não quer calar: "Por que colocaram isso em um livro sobre Python?"

A quem se destina este livro?

Se você puder responder "sim" a *todos* estes itens:

1. Você já sabe programar em outra linguagem de programação?
2. Quer adicionar o Python ao seu arsenal para programar com muita diversão como todos os outros programadores Python?
3. Você é o tipo de pessoa que aprende fazendo, preferindo fazer em vez de ouvir?

"Estilo Python": alguém que adora programar com Python.

Este livro é para você.

Quem provavelmente deve fugir deste livro?

Se puder responder "sim" a *qualquer um* destes itens:

1. Você nunca programou antes e não consegue identificar os *ifs* nos *loops*?
2. Você é um especialista em Python procurando um texto de referência conciso para a linguagem?
3. Você odeia aprender coisas novas? Acredita que nenhum livro sobre Python, não importa o quanto é bom, deve fazer você rir alto nem reclamar sobre como tudo é brega? Prefere o tédio às lágrimas?

Este livro não é para você.

[Nota do marketing: este livro é para qualquer pessoa com cartão de crédito.]

Sabemos o que você está pensando

"Como *isto* pode ser um livro sério sobre Python?"

"Por que tantas imagens?"

"Eu realmente posso *aprender* assim?"

Sabemos o que seu cérebro está pensando

Seu cérebro anseia por novidades. Está sempre buscando, examinando, *esperando* algo incomum. Foi construído dessa forma, e isso ajuda você a se manter vivo.

Então, o que seu cérebro faz com todas as coisas rotineiras, comuns e normais que você encontra? Tudo o que *pode* para impedi-las de interferir no trabalho *real* do cérebro — registrar as coisas que *importam*. Ele não se preocupa em gravar as coisas chatas; elas nunca conseguiriam passar no filtro "isto obviamente não é importante".

Como seu cérebro *sabe* o que é importante? Suponhamos que você esteja caminhando e um tigre pule na sua frente. O que acontece em sua cabeça e seu corpo?

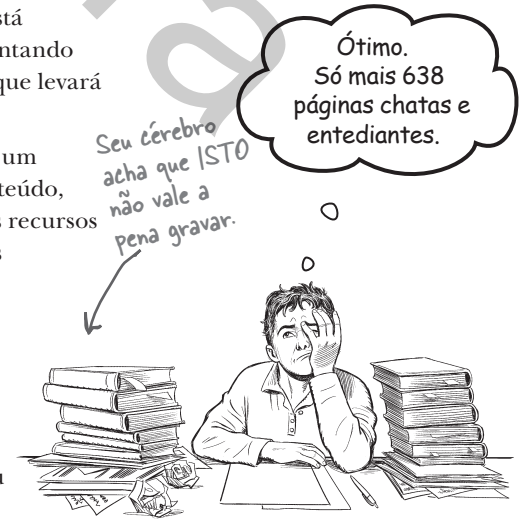
Os neurônios disparam. As emoções vão às alturas. Há uma *explosão química*.

E é assim que seu cérebro sabe…

Isto deve ser importante! Não se esqueça!

Mas imagine que você esteja em casa ou em uma biblioteca. É seguro, acolhedor, uma zona sem tigres. Você está estudando. Preparando-se para uma prova. Ou tentando aprender um assunto técnico que seu chefe acha que levará uma semana, dez dias no máximo.

Só um problema. Seu cérebro está tentando fazer um grande favor. Está tentando garantir que esse conteúdo, *obviamente* sem importância, não sobrecarregue os recursos escassos. Recursos que devem ser mais bem gastos armazenando coisas realmente *grandes*. Como tigres. Como o perigo de fogo. O fato de que você não deveria ter postado as fotos da "festa" nas redes sociais. E não há uma forma simples de dizer: "Ei, cérebro, muito obrigado, mas não importa a chatice deste livro e o pouco que estou registrando na escala Richter emocional agora, eu realmente quero que você grave essas coisas".



Achamos que o leitor "Use a Cabeça" é um aprendiz.

Então o que é necessário para *aprender* algo? Primeiro você tem que *entender*, então certificar-se de que não *esquecerá*. Não se trata de enfiar os fatos em sua cabeça. Com base nas pesquisas mais recentes em Ciência Cognitiva, Neurobiologia e Psicologia Educacional, a *aprendizagem* exige muito mais do que texto em uma página. Nós sabemos o que ativa seu cérebro.

Alguns princípios de aprendizagem do Use a Cabeça:

Torne-o visual. As imagens são muito mais inesquecíveis do que as palavras sozinhas e tornam a aprendizagem muito mais eficaz (até 89% de melhoria nos estudos de memória e transferência). Também torna as coisas mais compreensíveis. **Coloque as palavras dentro ou perto da imagem à qual se referem**, em vez de na parte inferior ou em outra página, e os alunos serão até *duas* vezes mais capazes de resolver os problemas relacionados ao conteúdo.

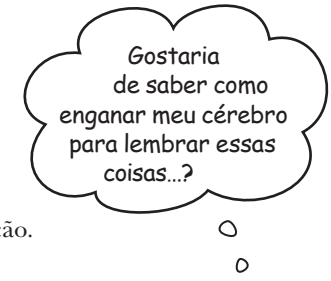
Use um estilo coloquial e personalizado. Em estudos recentes, os alunos se saíram até 40% melhor nos testes de pós-aprendizagem se o conteúdo falava diretamente com o leitor, usando um estilo coloquial em primeira pessoa, ao invés de adotar um tom formal. Conte histórias, em vez de dar palestras. Use uma linguagem informal. Não se leve muito a sério. Em que você prestaria mais atenção: em uma companhia estimulante no jantar ou em uma palestra?

Faça o aluno pensar mais profundamente. Em outras palavras, a menos que você movimente ativamente seus neurônios, nada acontecerá em sua cabeça. Um leitor tem que ser motivado, envolvido, ficar curioso e ser inspirado a resolver os problemas, tirar conclusões e gerar novos conhecimentos. E para isso você precisa de desafios, exercícios, perguntas instigantes e atividades que envolvam os dois lados do cérebro e vários sentidos.

Obtenha — e mantenha — a atenção do leitor. Todos já tivemos a experiência do "Eu realmente quero aprender, mas não consigo ficar acordado depois da página um". Seu cérebro presta atenção nas coisas incomuns, interessantes, estranhas, atraentes, inesperadas. Aprender um tópico novo, difícil e técnico não precisa ser chato. Seu cérebro aprenderá muito mais rapidamente se não for assim.

Toque suas emoções. Sabemos agora que a capacidade de se lembrar de algo depende muito do seu conteúdo emocional. Você se lembra daquilo que gosta. Você se lembra quando *sente* algo. Não, não estamos falando de histórias comoventes sobre um garoto e seu cão, mas de emoções como surpresa, curiosidade, diversão, "que diabos...?" e o sentimento de "eu domino!", quando você resolve um enigma, aprende algo que todo mundo acha difícil ou percebe que sabe algo que o Beto "sou mais técnico que você" da engenharia *não sabe*.

Metacognição: pensando sobre pensar



Se você realmente quer aprender, e quer aprender com mais rapidez e profundidade, preste atenção em como presta atenção. Pense em como você pensa. Aprenda como você aprende.

A maioria de nós não fez cursos sobre metacognição ou teoria da aprendizagem quando estava crescendo. Era *esperado* que aprendêssemos, mas raramente nos *ensinaram* a aprender.

Mas suponha que, se você está segurando este livro, realmente quer aprender a projetar sites de fácil utilização. E provavelmente não quer gastar muito tempo. Se quiser usar o que lê neste livro, será preciso se *lembrar* daquilo que lê. E para isso, tem que *entender*. Para tirar o máximo deste livro, de *qualquer* livro ou experiência de aprendizagem, seja responsável por seu cérebro. Seu cérebro *neste* conteúdo.

O truque é fazer com que o cérebro veja o novo material que você está aprendendo como algo Realmente Importante. Crucial para seu bem-estar. Tão importante quanto um tigre. Caso contrário, você estará em uma batalha constante, com o cérebro fazendo seu melhor para impedir a fixação do novo conteúdo.

Então como fazer com que seu cérebro trate o Python como se fosse um tigre faminto?

Há a maneira lenta e chata ou a forma mais rápida e eficaz. A forma lenta é a pura repetição. Obviamente, você saberá que *é* capaz de aprender e lembrar até o mais maçante dos tópicos se continuar batendo na mesma tecla em seu cérebro. Com repetição suficiente, o cérebro diz: "isto não *parece* importante, mas ele continua olhando a mesma coisa *sempre*, então acho que deve ser importante".

A forma mais rápida é fazer ***qualquer coisa que aumente a atividade do cérebro***, especialmente os *tipos* diferentes de atividade cerebral. As coisas na página anterior são uma grande parte da solução e todas comprovadamente ajudam o cérebro a trabalhar a seu favor. Por exemplo, estudos mostram que palavras colocadas *dentro* das imagens que descrevem (em oposição a algum outro lugar na página, como um título ou no corpo do texto) fazem com que seu cérebro tente entender como as palavras e as imagens se relacionam, e isso faz com que mais neurônios disparem. Mais neurônios disparando = mais chances de seu cérebro *perceber* que é algo digno de atenção e, possivelmente, de ser registrado.

Um estilo coloquial ajuda porque as pessoas tendem a prestar mais atenção quando percebem que estão em uma conversa, pois devem acompanhar e chegar até o fim. O surpreendente é que o cérebro não necessariamente se *importa* se a "conversa" é entre você e um livro! Por outro lado, se o estilo da escrita for formal e seco, o cérebro perceberá como se você estivesse assistindo a uma palestra, sentado em uma sala cheia de pessoas passivas. Não há necessidade de permanecer acordado.

Mas as imagens e o estilo coloquial são apenas o começo…

Veja O Que NÓS Fizemos:

Usamos ***imagens***, porque o cérebro se liga no visual, não no texto. Para seu cérebro, uma imagem *vale* por mil palavras. E quando texto e imagens trabalham juntos, incorporamos o texto *nas* imagens porque seu cérebro funciona com mais eficiência quando o texto está *dentro* daquilo a que ele se refere, ao contrário de um título ou oculto em algum lugar do texto.

Utilizamos a ***redundância***, dizendo a mesma coisa de formas *diferentes*, com diferentes tipos de mídia e *vários sentidos*, para aumentar as chances de que o conteúdo fique codificado em mais de uma área do seu cérebro.

Usamos conceitos e imagens de formas ***inesperadas***, porque seu cérebro gosta de novidade, e usamos as imagens e as ideias com pelo menos *algum conteúdo* ***emocional***, porque o cérebro está ligado para prestar atenção na bioquímica das emoções. É mais provável que o que o faz *sentir* algo seja lembrado, mesmo que esse sentimento não seja nada mais do que um pouco de ***humor***, ***surpresa*** ou ***interesse.***

Usamos um ***estilo coloquial*** personalizado, porque seu cérebro presta mais atenção quando acredita que você está em uma conversa do que quando está passivo ouvindo uma apresentação. O cérebro faz isso mesmo quando você está *lendo*.

Incluímos mais de 80 ***atividades***, porque seu cérebro aprende e lembra mais quando você ***faz*** as coisas do que quando *lê* sobre elas. E fizemos exercícios desafiadores, porque é isso que a maioria das pessoas prefere.

Usamos ***vários estilos de aprendizagem***, porque você pode preferir procedimentos passo a passo, enquanto alguém deseja compreender a imagem geral primeiro e uma terceira pessoa só quer ver um exemplo. Mas, independentemente de sua preferência de aprendizagem, *todos* se beneficiam vendo o mesmo conteúdo representado de várias maneiras.

Incluímos conteúdo para os ***dois lados do cérebro***, porque, quanto mais seu cérebro se envolver, mais você aprenderá e lembrará, e poderá manter o foco por mais tempo. Como trabalhar com um lado do cérebro muitas vezes significa dar ao outro lado a oportunidade de descansar, você poderá ser mais produtivo na aprendizagem por um período de tempo maior.

E incluímos ***histórias*** e exercícios que apresentam ***mais de um ponto de vista***, porque seu cérebro aprende mais profundamente quando é forçado a fazer avaliações e julgamentos.

Foram incluídos ***desafios***, com os exercícios, fazendo ***perguntas*** para as quais nem sempre há uma resposta certa, porque seu cérebro aprende e lembra quando tem que *trabalhar* em algo. Pense — você não pode *colocar* seu corpo em forma apenas observando as pessoas na academia. Mas fizemos o melhor para ter certeza de que, quando você estiver trabalhando pesado, será nas coisas *certas*. Que ***você não estará gastando um dendrito a mais*** processando um exemplo complicado de entender, analisando um jargão difícil ou um texto muito conciso.

Usamos ***pessoas*** nas histórias, nos exemplos, nas imagens etc. porque, bem, *você* é uma pessoa. E seu cérebro presta mais atenção nas pessoas do que nas *coisas*.

Veja o que VOCÊ pode fazer para o cérebro se submeter

Fizemos a nossa parte. O resto é com você. Estas dicas são um começo; ouça seu cérebro e descubra o que funciona ou não para você. Experimente coisas novas.

Recorte isto e cole em sua geladeira.

1 Vá devagar. Quanto mais você entende, menos tem que memorizar.

Não *leia* apenas. Pare e pense. Quando o livro fizer uma pergunta, não pule para a resposta. Imagine que alguém realmente *esteja* fazendo a pergunta. Quanto mais profundamente você forçar seu cérebro a pensar, mais chances terá de aprender e lembrar.

2 Faça os exercícios. Faça suas próprias anotações.

Nós os colocamos, mas se fizéssemos por você, seria como se alguém fizesse seus exercícios. E não *olhe* apenas os exercícios. **Use um lápis.** Há muitas evidências de que a atividade física *ao* aprender pode aumentar o aprendizado.

3 Leia as seções "Não existem perguntas idiotas".

Isto significa todas elas. Elas não são seções opcionais, ***fazem parte do conteúdo principal!*** Não as pule.

4 Que seja a última coisa que você lê antes de dormir. Ou pelo menos a última coisa desafiadora.

Parte da aprendizagem (principalmente a transferência para a memória de longo prazo) acontece *depois* que você fecha o livro. Seu cérebro precisa de tempo para fazer mais processamento. Se você colocar algo novo durante esse tempo de processamento, algumas coisas que aprendeu serão perdidas.

5 Fale sobre isso. Em voz alta.

Falar ativa uma parte diferente do cérebro. Se você tenta entender algo ou aumentar sua chance de lembrar mais tarde, diga em voz alta. Melhor ainda, tente explicar em voz alta para alguém. Você aprenderá mais rapidamente e poderá descobrir ideias que não sabia que existiam quando estava lendo sobre o assunto.

6 Beba água. Em grande quantidade.

Seu cérebro funciona melhor em uma bela banheira de líquido. A desidratação (que pode acontecer antes de você sentir sede) diminui a função cognitiva.

7 Ouça seu cérebro.

Preste atenção se seu cérebro está ficando sobrecarregado. Se você está começando a passar os olhos ou esquece o que acabou de ler, é hora de uma pausa. Assim que passar de certo ponto, não aprenderá mais rapidamente tentando colocar mais informações, e pode até prejudicar o processo.

8 Sinta algo.

Seu cérebro precisa saber que isso *significa algo*. Envolva-se com as histórias. Crie seus próprios títulos para as fotos. Lamentar uma piada de mau gosto é *ainda* melhor do que não sentir nada.

9 Escreva muito código!

Há apenas uma forma de aprender a programar em Python: **escrevendo muito código**. E é isso que você fará neste livro. A codificação é uma habilidade, e a única forma de ser bom nisso é praticando. Daremos a você muita prática: todo capítulo tem exercícios que propõem um problema a ser resolvido. Não os pule simplesmente — muito da aprendizagem acontece quando você resolve os exercícios. Incluímos uma solução para cada exercício — não tenha medo de **espiar a solução** se surgir uma dúvida! (É fácil ter problemas com coisas pequenas.) Mas tente resolver o problema antes de olhar a solução. E, definitivamente, faça com que funcione antes de passar para a próxima parte do livro.

Leia-me

É uma experiência de aprendizagem, não um livro de consulta. Retiramos deliberadamente tudo o que poderia atrapalhar a aprendizagem do que quer que estejamos aprendendo nesse ponto do livro. E na primeira vez você precisará iniciar no começo, porque o livro faz suposições sobre o que já viu e aprendeu.

Este livro é planejado para você entender o mais rápido possível.

À medida que você precisa saber das coisas, ensinamos. Então você não encontrará longas listas de material técnico, nenhuma tabela de operadores do Python, nem de suas regras de precedência de operadores. Não cobrimos *tudo*, mas trabalhamos muito para cobrir o material indispensável, assim você pode colocar o Python em seu cérebro *rapidamente* e fazer com que fique lá. A única suposição que fazemos é que você já sabe programar em alguma linguagem de programação.

Você pode usar com segurança qualquer versão do Python 3.

Bem… talvez seja uma mentirinha inocente. Você precisa *pelo menos* do Python 3.6, mas como essa versão apareceu pela primeira vez no final de 2016, é improvável que muitas pessoas dependam dela no dia a dia. Só para constar: no momento da impressão, o Python 3.12 estava chegando.

As atividades NÃO são opcionais.

Os exercícios e as atividades não são complementos; fazem parte do conteúdo básico do livro. Alguns são para ajudar a memória, outros são para a compreensão e há aqueles que ajudarão a aplicar o que você aprendeu. ***Não pule os exercícios.***

No final de cada capítulo, você encontrará uma palavra cruzada criada para testar sua retenção do material apresentado até o momento. Todas as respostas para as pistas são do capítulo recém-concluído. As soluções são fornecidas na última página de cada capítulo. Tente completar cada palavra cruzada sem espiar. Dito isso, as palavras cruzadas são a única coisa que você não *precisa* fazer, mas são boas para dar ao cérebro uma chance de pensar sobre as palavras e os termos que você aprendeu em um contexto diferente.

A redundância é intencional e importante.

Uma clara diferença em um livro *Use a Cabeça* é que queremos que você *realmente* entenda. E queremos que termine o livro lembrando-se daquilo que aprendeu. A maioria dos livros de consulta não tem fixação e recordação como meta, mas este aqui é voltado para a *aprendizagem*, portanto, você verá alguns dos mesmos conceitos aparecendo mais de uma vez. É de propósito.

Os exemplos são os mais enxutos possíveis.

Nossos leitores dizem que é frustrante percorrer 200 linhas de um exemplo procurando as duas linhas que eles precisam entender. A maioria dos exemplos neste livro é apresentada dentro do menor contexto possível, para que a parte que você está tentando aprender seja clara e simples. Não espere que todos os exemplos sejam robustos ou mesmo completos — eles são escritos especificamente para o aprendizado e nem sempre são totalmente funcionais (embora tentemos assegurar ao máximo que sejam).

Colocamos todos os códigos e os recursos deste livro na web para que você possa usá-los quando necessário. Dito isso, acreditamos que há muito a aprender ao digitar o código à medida que você *acompanha*, sobretudo quando aprende uma nova linguagem de programação (para você). Mas para as pessoas do tipo "só quero o código", aqui está a página GitHub do livro [em inglês, assim como todos os outros links]:

https://github.com/headfirstpython/third

Os exercícios Poder do Cérebro não têm respostas.

Para alguns, não há uma resposta certa e para outros, parte da experiência de aprendizagem das atividades *Poder do Cérebro* é para você decidir se e quando suas respostas estão certas.

Nem todos os exercícios Test Drive têm respostas.

Para alguns exercícios, simplesmente pedimos que você siga um conjunto de instruções. Nós lhe daremos formas de verificar se o que você fez realmente funcionou, mas ao contrário dos outros exercícios, não há respostas certas!

Instale o Python mais recente

O que você faz aqui depende da plataforma que está executando, que é supostamente *Windows*, *macOS* ou *Linux*. A boa notícia é que todas as três plataformas rodam o Python mais recente. Não há más notícias.

Se já estiver executando a versão 3 do Python, vá para a próxima página — você está pronto. Se ainda não instalou o Python ou está usando uma versão mais antiga, selecione o parágrafo seguinte que se aplica e continue lendo.

Instalando no Windows

O pessoal maravilhoso do Python na Microsoft trabalha muito para garantir que a versão mais recente do Python esteja sempre disponível para você via aplicação *Windows Store*. Abra-a, procure por "Python", selecione a versão mais recente e clique no botão **Obter**. Observe pacientemente enquanto o indicador de progresso se move de zero a 100%, em seguida, com a instalação concluída, vá para a próxima página — você está pronto.

Se você estiver rodando o Windows 11 ou posterior, também pode querer baixar o novo Terminal do Windows, com uma experiência de linha de comando melhor/mais agradável (se precisar).

Instalando no macOS

Os Macs mais recentes tendem a ser enviados com versões um pouco mais antigas do Python. *Não as use*. Vá para a página inicial do Python na web, *https://www.python.org*, e clique na opção "Downloads". A última versão do Python 3 deve começar a baixar, pois o site Python é inteligente o suficiente para detectar que você está se conectando a partir de um Mac. Assim que o download concluir, execute o instalador que está esperando por você na sua pasta Downloads. Clique no botão **Próximo** até que não haja mais botões **Próximo** para clicar, depois, quando a instalação terminar, vá para a próxima página — você está pronto.

Não é preciso remover as versões pré-instaladas mais antigas do Python que vêm com o seu Mac. Esta instalação irá substituí-las.

Uma alternativa é que se você é um usuário do *Gerenciador de Pacotes Homebrew*, está com sorte, pois o Homebrew tem suporte para download e instalação da última versão do Python.

Instalando no Linux

Programadores Use a Cabeça são uma equipe improvisada de técnicos cujo trabalho é manter os *Autores Use a Cabeça* nos trilhos (um feito notável). Os programadores adoram o *Linux* e a distribuição *Ubuntu*, e isso é visto aqui.

Não é nenhuma surpresa que o Ubuntu mais recente venha com o Python 3 instalado e atualizado. Se esse for o caso, legal, você está pronto, embora possa querer usar o **apt** para instalar o `pacote python3-pip` também.

Se estiver usando uma distribuição Linux diferente do Ubuntu, use o gerenciador de pacotes do seu sistema para instalar o Python 3 no Linux. Uma vez feito, vá para a próxima página — você está pronto.

O Python apenas não é suficiente

Completaremos a instalação com duas coisas: uma dependência de back-end necessária, bem como um editor de texto moderno e compatível com o Python. Para explorar, experimentar e aprender sobre Python, você precisa instalar um back-end de execução chamado *Jupyter* no Python. Como verá em breve, fazer isso é simples. Quando se trata de criar código Python, você pode usar *qualquer* editor do programador, mas recomendamos usar um específico ao trabalhar com o material deste livro: *Visual Studio Code* da Microsoft, conhecido no mundo inteiro como **VS Code**.

Instale o back-end mais novo do Jupyter Notebook

Não se preocupe, você aprenderá tudo sobre seu uso em breve!

Independentemente do sistema operacional sendo rodado, verifique se você está conectado à internet, abra uma janela de terminal e digite:

```
python3 -m pip install jupyter
```

Em alguns sistemas, o executável que roda seu código Python é chamado "python"; em outros se chama "python3". Pode ser difícil acompanhar. 😉

Uma série de mensagens de status passa pela tela. Se você estiver vendo uma mensagem quase no fim dizendo que tudo está "Instalado com sucesso", então tudo bem. Se não, verifique os documentos do Jupyter e tente novamente.

Instale a versão mais recente do VS Code

Existem alternativas, mas — em nossa visão — o VS Code é imbatível quando se trata do material deste livro. E, não, nós *não* fazemos parte de uma conspiração global para promover os produtos da Microsoft!!

Pegue seu navegador favorito e navegue até a página de download do VS Code:

https://code.visualstudio.com/Download

Escolha o download que corresponde ao seu ambiente e aguarde a conclusão dele. Siga as instruções do site para instalar o VS Code, em seguida, vire a página para aprender como concluir a configuração dele.

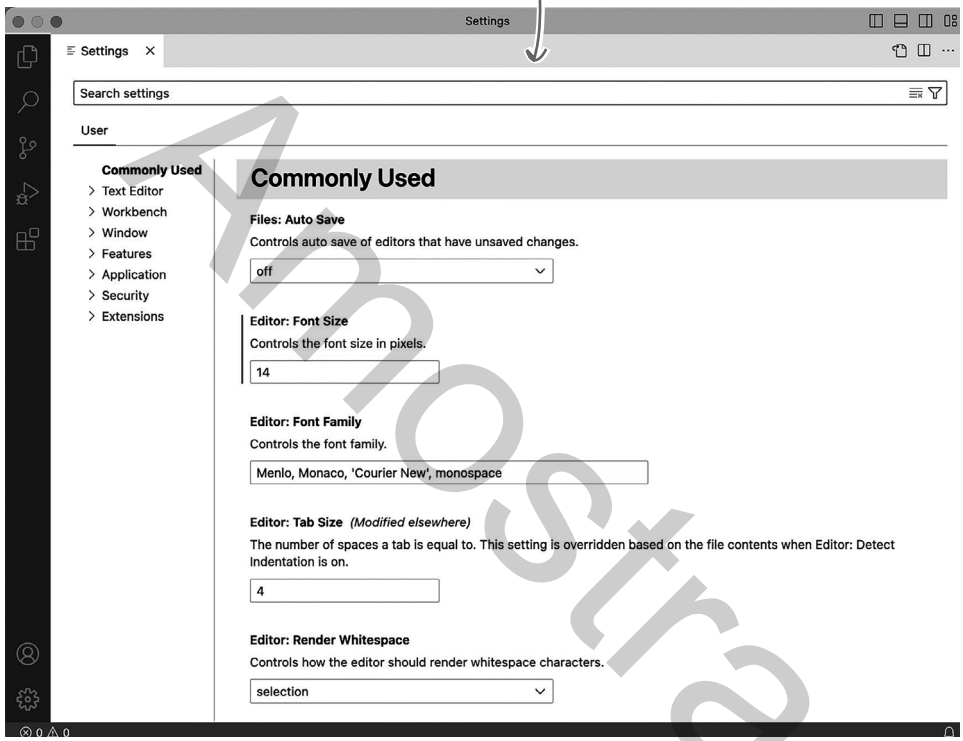
Configure o VS Code como quiser

Vá em frente e execute o VS Code pela primeira vez. No menu, selecione **Arquivo**, **Preferências** e **Configurações** para acessar as preferências de configuração do editor.

No Mac, comece com o menu "Código".

Você deve ver algo assim:

Pesquise qualquer configuração digitando parte ou todo o seu nome nesta caixa de pesquisa..



Até se familiarizar com o VS Code, você pode configurar seu editor para corresponder às configurações preferidas pelos *Programadores Use a Cabeça*. Veja as configurações usadas neste livro:

Você não precisa usar essas configurações, mas se quiser combinar sua tela com o que vê neste livro, então estes ajustes são recomendados.

- As *guias de indentação* estão desativadas.
- O *tema de cores* do editor está definido para **Claro**.
- O *minimapa* do editor está desativado.
- O *destaque de ocorrências* do editor está desativado.
- O *destaque da linha de renderização* do editor está definido para **nenhum**.
- O *tamanho da fonte* do terminal e do editor de texto está definido para **14**.
- A *barra de status da célula de exibição* do notebook está definida para **oculta**.
- A *lâmpada* do editor está desativada.

Adicione duas extensões requeridas ao VS Code

Você não é obrigado a copiar a configuração recomendada do editor, mas deve instalar duas extensões do VS Code: **Python** e **Jupyter**.

Quando terminar de ajustar suas configurações preferidas do editor, feche a guia Configurações clicando no X. Em seguida, para pesquisar, selecionar e instalar as extensões, clique no ícone Extensões à esquerda da tela principal do VS Code:

A lista de extensões instaladas está vazia inicialmente. Mas — veja! — o VS Code faz algumas recomendações.

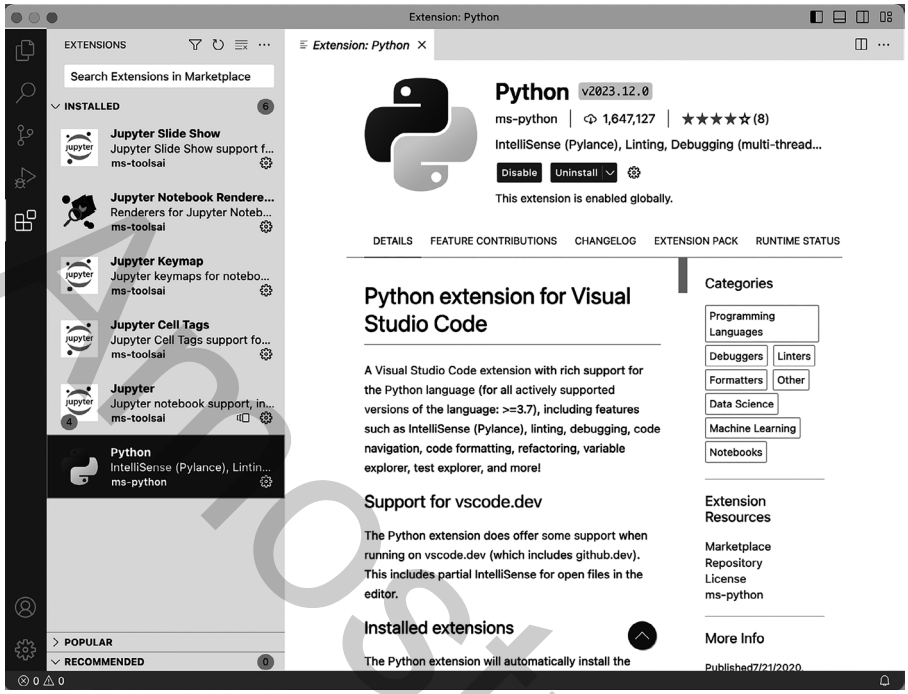
Clique no ícone Extensões para deslizar o painel. Você pode pesquisar, selecionar e instalar as extensões daqui.

Se estiver vendo "Python" e "Jupyter" em sua lista, clique no botão Instalar para adicionar cada extensão ao VS Code. Se não estiver vendo essas recomendações, use a caixa de pesquisa para encontrá-las e fazer a instalação.

O suporte ao Python do VS Code é avançado

Instalar as extensões Python e Jupyter realmente resulta em algumas instalações adicionais de extensão do VS Code, como mostrado aqui:

Se você esperava apenas duas extensões na lista de instaladas, pode se surpreender ao ver mais algumas. Não se preocupe..



Estas extensões adicionais melhoram o suporte do VS Code para Python e Jupyter além do que está incluído nas extensões padrão. Embora você não precise saber o que elas fazem (por enquanto), saiba isto: elas ajudam a tornar o VS Code um editor Python *turbinado*.

Com Python 3, Jupyter e VS Code instalados, você está preparado!

Se Liga!!

Ao longo do livro você encontrará boxes técnicos como este. Os boxes *Se Liga!* são usados para detalhar mais um tópico específico do que normalmente fazemos no texto principal. Não entre em pânico se o material nesses boxes tirar seu foco. Eles são planejados para acalmar seu nerd interior curioso. Você pode pular com segurança (e sem culpa) qualquer Se Liga! em uma primeira leitura.

Sim, é um Se Liga! sobre os boxes Se Liga! (e não teremos nenhuma piada de recursão constrangedora, obrigado).

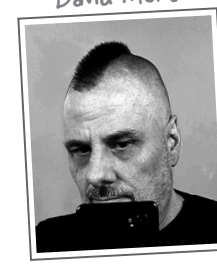
Equipe de Revisão Técnica

Daniel Hinojosa

Mike Hopkins

Dave Marsden

David Mertz



William Jamir Silva

Conheça a Equipe de Revisão Técnica do *Use a Cabeça Python*

A Equipe teve a oportunidade de revisar o rascunho de Lançamento Antecipado deste livro e, ao fazê-lo, contribuiu para tornar o artefato final publicado muito melhor. Cada comentário foi lido, considerado, então decidido ou posto em prática. Eles ajudaram a corrigir os códigos corrompidos, melhoraram muitas de nossas explicações e — em alguns lugares — forneceram melhorias de código para alguns dos nossos erros de codificação embaraçosos.

Nem é preciso dizer que tudo o que ainda está errado é nossa culpa, e só nossa.

Os nossos sinceros agradecimentos a cada um dos revisores: o trabalho de vocês é muito valorizado.

[E ao misterioso *Mark*, o revisor de tecnologia que preferiu o anonimato: obrigado também por suas excelentes sugestões.]

Agradecimentos

Melissa Potter

Na O'Reilly, começarei com **Melissa Potter**, editora deste livro.

Melissa foi um trunfo para este livro, embora (às vezes) ela possa ter desanimado quando as coisas demoraram muito. Sou muito grato por sua orientação e paciência em todo o processo, mas especialmente por seu olhar atento quando se tratava de detectar problemas no *fluxo* do livro e no meu uso da linguagem. Eu sempre disse que, independentemente de você estar escrevendo um livro ou não, t*odo mundo precisa de um editor*, e estou muito feliz que a minha tenha sido Melissa para a terceira edição.

Da perspectiva da série *Use a Cabeça*, agradeço a **Beth Robson**, uma das consultoras, por fazer comentários detalhados e dar sugestões sobre o material de Lançamento Antecipado. Espero que o livro pronto faça Beth sorrir.

Há muitas pessoas nos bastidores da O'Reilly. Meus agradecimentos a todas, mas sobretudo à equipe de **Produção** que pegou minhas (ainda amadoras) páginas do InDesign e as transformou em um lindo livro impresso. Obrigado.

No trabalho, meus agradecimentos novamente ao meu chefe de departamento **Nigel Whyte** por sempre incentivar meu envolvimento nesses projetos de escrita. Obrigado também a meus alunos de **Desenvolvimento de Jogos** e **Desenvolvimento de Software**, bem como a meus pós-graduados em **Ciência de Dados —** ao longo dos anos versões deste material foram impostas a eles. É sempre um prazer ensinar a esses grupos, e vê-los reagir positivamente ao aprender a linguagem Python me faz um bem enorme.

Em casa, minha esposa **Deirdre** teve que suportar mais um projeto de escrita que, como antes, parecia consumir cada momento acordado. Nossos filhos estavam na adolescência (ou eram mais novos) quando a primeira edição deste livro foi escrita e, agora, estão todos na faculdade e buscando coisas maiores e melhores. Nem preciso dizer: eu estaria perdido sem o apoio e o amor da minha esposa e dos meus filhos.

Uma nota rápida sobre as palavras cruzadas no final dos capítulos: elas foram geradas pelo maravilhoso programa `genxword` de **David Whitlock** (com alguns ajustes locais de Paul). `genxword` é escrito em Python e está, claro, disponível para download no PyPI.

0 por que Python?

Parecido, mas Diferente

O Python começa contando do zero, o que deve ser familiar.

Na verdade, o Python tem muito em **comum** com outras linguagens de programação. Existem **variáveis**, **loops**, **condicionais**, **funções** e afins. Neste capítulo de abertura, faremos um **tour avançado e rápido** pelos conceitos básicos do Python, apresentando a linguagem sem detalhar muito. Você aprenderá a **criar** e **executar** código com o Jupyter Notebook (rodando no VS Code). Verá como muitas funcionalidades de programação vêm **predefinidas** no Python, que você **utilizará** para fazer as coisas. Também aprenderá que, embora o Python compartilhe muitas de suas ideias com outras linguagens de programação, como elas se manifestam em seu código pode ser bem **diferente**. Agora, não entenda mal: estamos falando sobre algo **bom** e diferente, não sobre uma diferença *ruim*. Leia para saber mais…

Python – A revelação

A entrevista de hoje é com um convidado muito especial: **a linguagem de programação Python**.

Use a Cabeça: Deixe-me apenas dizer que é um prazer tê-lo aqui hoje.

Python: Obrigado.

Use a Cabeça: Você é considerado uma das linguagens de programação mais populares em uso hoje. Por que você acha que é?

Python: (Ficando corado) Meu… isso é realmente um elogio. Imagino que muitos programadores gostam de mim.

Use a Cabeça: Claro. Mas por que você acha que é? O que torna *você* tão diferente?

Python: Não acho que sou tão diferente da maioria das outras linguagens de programação. Tenho variáveis, loops, condicionais, funções etc. como todas as outras.

Use a Cabeça: OK, mas o que explica seu sucesso?

Python: Eu acho que é uma combinação de coisas.

Use a Cabeça: Como…?

Python: Bem… Dizem que sou fácil de ler.

Use a Cabeça: Você quer dizer que as pessoas sabem o que você está pensando?!?

Python: Ah, é muito engraçado e hilário. Claro, me refiro ao fato de que meu *código* é fácil de ler.

Use a Cabeça: E por que você acha isso?

Python: Para começar, eu não preciso de ponto e vírgula no final das minhas instruções e parênteses em *tudo* é menos necessário do que em outras linguagens. Isso torna meu código muito limpo, o que ajuda na legibilidade e na compreensão.

Use a Cabeça: O que mais?

Python: Bem, tem a indentação. Na minha humilde opinião, eu recebo um monte de críticas *injustificadas* por isso, pois uso a indentação com espaço em branco para sinalizar os blocos de código. Isso faz meu código parecer consistente, limpo e legível. E mais, não há discussão sobre a colocação "correta" das chaves.

Use a Cabeça: Então… sem chaves no código?

Python: Na verdade, existem sim, mas não nos blocos. No Python, as chaves ficam nos dados, *não* no código. Para indicar os blocos, recuo com espaços em branco.

Use a Cabeça: Interessante… e imagino que os programadores habituados a linguagens como C, C++, Java, C#, Go, Rust etc., têm problemas com isso.

Python: Sim, infelizmente velhos hábitos são difíceis de abandonar. Agora, eu não estou apenas dizendo isso, mas depois de escrever código comigo, você quase nunca notará o espaço em branco, pois ele se torna natural muito rapidamente. É bom também que os editores modernos sejam versados em Python, fazendo a indentação para você. Raramente há motivos para reclamar.

Use a Cabeça: Com certeza a legibilidade é *Algo Bom*™, mas deve haver mais para sua popularidade?

Python: Há também minha Biblioteca Padrão: uma coleção de bibliotecas de código incluídas que ajudam em todos os problemas de programação. E, claro, há o PyPI?

Use a Cabeça: Py…P… o quê?!?

Python: Py…P…I, uma abreviação de *Python Package Index*, um repositório online, globalmente acessível, de módulos Python compartilhados de terceiros. Eles ajudam os programadores a resolver todos os problemas de programação imagináveis. É um recurso maravilhoso para a comunidade.

Use a Cabeça: Então esse tal PyPI permite explorar o código de outro programador *sem* escrever o seu próprio?

Python: Parece meio sórdido quando você coloca assim…

A entrevista continua na próxima página. →

Use a Cabeça: Desculpe, talvez "utilizar" seja uma palavra melhor?

Python: Sim, isso mesmo: *utilizar*. Minha filosofia é garantir que os programadores só escrevam código novo quando realmente precisam.

Use a Cabeça: Como assim?

Python: Uma palavra: *definido previamente*.

Use a Cabeça: São duas palavras…

Python: Sim, mas significa predefinido… de qualquer forma, o resultado é que eu tenho mais do que minha Biblioteca Padrão predefinida…

Use a Cabeça: … Continue…

Python: Estou falando sobre minhas *outras* predefinições. Para começar, há as minhas funções predefinidas (BIFs). São pequenos motores de funcionalidade genérica que operam em muitos lugares e situações.

Use a Cabeça: Dê aos nossos leitores um exemplo rápido.

Python: Considere **len**, uma abreviação de "length" (comprimento). Dê um objeto a **len** e ela informa o tamanho dele.

Use a Cabeça: Ainda bem que estou sentado.

Python: Ha ha. Eu sei que parece simples, mas **len** é uma maravilha. Dê uma lista a **len** e ela diz quantos objetos existem nela. Dê uma string e **len** diz quantos caracteres existem etc. Se um objeto pode informar seu tamanho, **len** diz o que é.

Use a Cabeça: Então **len** é *polimórfica*?

Python: Epa, é uma linguagem chique, né?? 😉 Mas, sim, como muitas das minhas BIFs, **len** trabalha com muitos objetos diferentes de muitos tipos diferentes. Em último caso, eu concordaria que **len** é polimórfica.

Use a Cabeça: O que mais torna você popular?

Python: Minha personalidade envolvente.

Use a Cabeça: Tem mais a ver com sua personalidade *despreocupada*. Vamos manter o tom profissional.

Python: Certo, posso me divertir um pouco? Afinal, meu nome vem de um grupo britânico de comédia dos anos 1960.

Use a Cabeça: Sério? Agora você está me zoando.

Python: É verdade. Pesquise (veja: *https://docs.python.org/3/faq/general.html#why-is-it-called-python*). Meu nome *não* vem de uma cobra. Ele vem da *série de comédia Monty Python*.

Use a Cabeça: Se você diz…

Python: E sou fã de enlatados para *tudo*.

Use a Cabeça: OK, agora você está sendo bobo.

Python: Certo, prometo me comportar. Vá em frente, faça outra pergunta.

Use a Cabeça: Qual é seu lance com os dados?

Python: Eu simplesmente *amo* dados, trabalhando com minhas estruturas de dados predefinidas (são realmente legais), falando com bancos de dados ou buscando dados na web, sou sua linguagem de programação ideal para a manipulação de dados.

Use a Cabeça: Acho que seu amor pelos dados reflete sua primeira posição no mundo do aprendizado de máquina, né?

Python: Acho que sim. Para ser honesto, eu acho toda essa coisa de ML um pouco extravagante. Estou igualmente feliz por executar suas funções, acabar com seus loops e ajudá-lo a trabalhar.

Use a Cabeça: Mas certamente a fusão do Python, dos dados e do DL é uma combinação dos deuses!

Python: Sim, eu acho. Mas nem todos precisam fazer esse tipo de coisa. Não me interprete mal, estou emocionado por ser um influencer no campo da IA, mas não é *tudo* o que faço.

Use a Cabeça: Explique.

Python: Claro. Em essência, sou uma linguagem de programação geral, que pode ter muitos usos. Fico igualmente feliz por executar seu webapp mais recente e por rodar no *Mars Rover*. É só código para mim. Um *código* lindamente formatado, fácil de ler. É disso que se trata.

Use a Cabeça: Concordo plenamente. Obrigado, Python, por conversar conosco hoje.

Python: Por nada. Até!

Olá. Tenho um projeto para você que me foi passado como sendo perfeito para o Python. O problema é que, como você, sou nova em Python. Eu gostaria de saber se há um modo rápido de ter uma ideia antes de entrar de cabeça no projeto.

Claro. Vamos conhecer o Python.

Iremos esperar para falar sobre o projeto no próximo capítulo. Agora, nos concentraremos em chegar ao ponto em que você tem uma boa compreensão de alguns conceitos básicos, como usar suas ferramentas para **criar**, **editar** e **executar** o código Python. Também apresentaremos alguns recursos da linguagem Python para dar a *ideia* de que precisa. Manteremos o exemplo simples, usando-o principalmente para apresentar uma visão geral do Python em vez de usá-lo para resolver um problema específico (haverá muito *disso* em breve).

Como a entrevista com o Python confirma, há várias razões para a popularidade dele. Listamos as conclusões que obtivemos da entrevista no final desta página.

Passaremos um tempo detalhando mais esses pontos. Quando tiver examinado a lista, pegue um lápis — sim, um lápis — e encontre-nos no topo da próxima página!

E não se preocupe: tudo mencionado na visão geral deste capítulo é coberto em detalhes mais adiante no livro.

1. O código Python é fácil de ler.
2. O Python vem com uma Biblioteca Padrão.
3. O Python tem funções predefinidas práticas, poderosas e genéricas.
4. O Python vem com estruturas de dados predefinidas.
5. O Python tem Python Package Index (PyPI).
6. O Python não se leva muito a sério.

Não subestime a importância deste último ponto.

Veja como é fácil ler em Python

Você ainda não sabe muito sobre Python, mas apostamos que pode dar bons palpites sobre como o código Python funciona. Dê uma olhada em cada linha de código abaixo e escreva o que acha que ela faz. Fizemos a primeira para você começar. Não se preocupe se ainda não entende todo o código — as respostas estão na próxima página, então fique à vontade para dar uma espiada se tiver dificuldades.

```
import random

suits = ["Clubs", "Spades", "Hearts", "Diamonds"]

faces = ["Jack", "Queen", "King", "Ace"]

numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]


def draw():

   the_suit = random.choice(suits)

   the_card = random.choice(faces + numbered)

   return the_card, "of", the_suit


print(draw())

print(draw())

print(draw())
```

Uma biblioteca é incluída para dar suporte para a aleatoriedade.

Veja como é fácil ler em Python

Você ainda não sabe muito sobre Python, mas apostamos que pode dar bons palpites sobre como o código Python funciona. Você deveria escrever o que achou que cada linha do código abaixo fez. Fizemos a primeira para você começar. Como suas anotações se comparam com as nossas?

`import random`	Uma biblioteca é incluída para dar suporte para a aleatoriedade.
`suits = ["Clubs", "Spades", "Hearts", "Diamonds"]`	Uma variável "suits" é atribuída a um array de 4 strings, representando os naipes em um baralho de cartas.
`faces = ["Jack", "Queen", "King", "Ace"]`	Outra variável "faces" é atribuída a um array com as 4 cartas de figura.
`numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]`	E aqui está outra variável de array "numbered", que representa as cartas numeradas do baralho.
`def draw():`	Parece o início de uma função.
`the_suit = random.choice(suits)`	Seleciona aleatoriamente um naipe e o atribui a uma nova variável "the_suit".
`the_card = random.choice(faces + numbered)`	Seleciona aleatoriamente uma carta a partir da combinação dos arrays "faces" e "numbered", criando "the_card."
`return the_card, "of", the_suit`	Retorna a carta selecionada e uma string contendo a palavra "of", em seguida, o naipe selecionado.
`print(draw())`	Chama a função "draw" para pegar uma carta do baralho e mostra a carta na tela com "print".
`print(draw())`	Idem
`print(draw())`	Idem

Pode parecer um pouco estranho, pois o código chama a função "draw" primeiro, retornando a carta aleatória, que é então passada para "print" (outra função), que gera a saída.

Pronto para rodar algum código

Há um pouco de limpeza a fazer *antes* de executar qualquer código.

Para manter as coisas organizadas, criaremos uma pasta no seu computador chamada *Aprendizagem*. Você pode colocar essa pasta em qualquer lugar do seu disco rígido, desde que se lembre onde foi, pois irá usá-la *o tempo todo*. [Se precisar, coloque a nova pasta *Aprendizagem* dentro de uma pasta *UseaCabeça* para diferenciá-la de qualquer outra pasta *Aprendizagem* que você possa ter.]

Colocaremos tudo a fazer com este livro aqui.

Com a nova pasta *Aprendizagem* criada, inicie o VS Code.

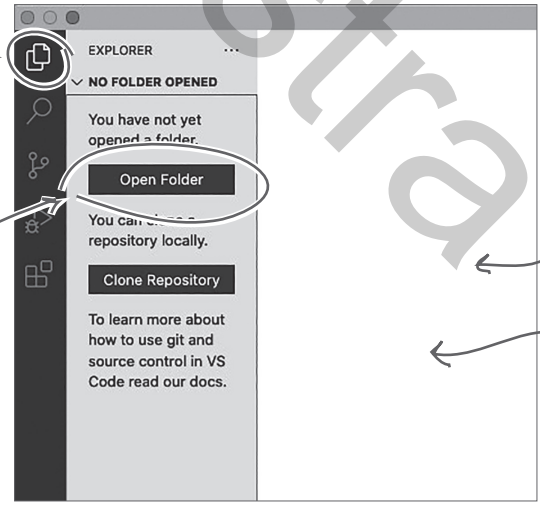
Relaxe

Nada de pânico se você não instalou o VS Code. Não é grande coisa. Volte à *Introdução* deste livro e trabalhe nas páginas que começam em **Instale o Python mais recente**. Você encontrará instruções para instalar o VS Code e algumas extensões necessárias lá.

Faça isso agora e volte aqui quando estiver pronto. Vamos esperar…

Não fique mal se pulou a introdução. Você não é o primeiro a fazer isso, e não será o último. 😉

Quando o VS Code inicia, ele normalmente leva você de volta ao que estava trabalhando antes ou você verá a página "Iniciar". De qualquer forma, feche a(s) página(s) aberta(s) do editor e clique no primeiro ícone à esquerda superior para abrir o painel Explorador.



Clique neste botão para selecionar e abrir sua pasta "Aprendizagem".

Não mostramos a exibição completa do VS Code aqui porque está vazia.

Em alguns sistemas, o VS Code pode pedir que você confirme se confia nos criadores da pasta na qual está trabalhando... você confia em si mesmo, não é?

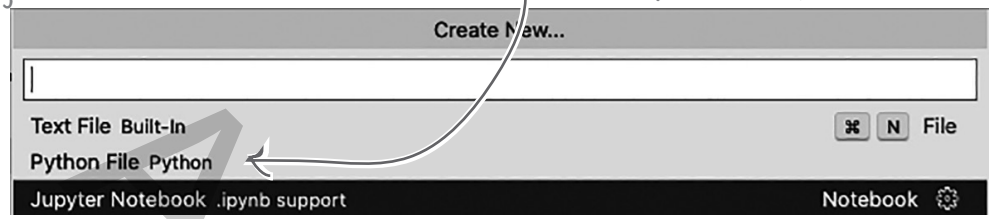
Sempre que você trabalhar com o VS Code neste livro, abrirá sua pasta *Aprendizagem* quando necessário. **Faça isso agora antes de continuar**.

Sua primeira experiência com Jupyter

OK. Você está rodando o VS Code e abriu sua pasta *Aprendizagem*. Criaremos um novo notebook selecionando primeiro o menu **Arquivo** e a opção **Novo arquivo...**. Você verá três opções:

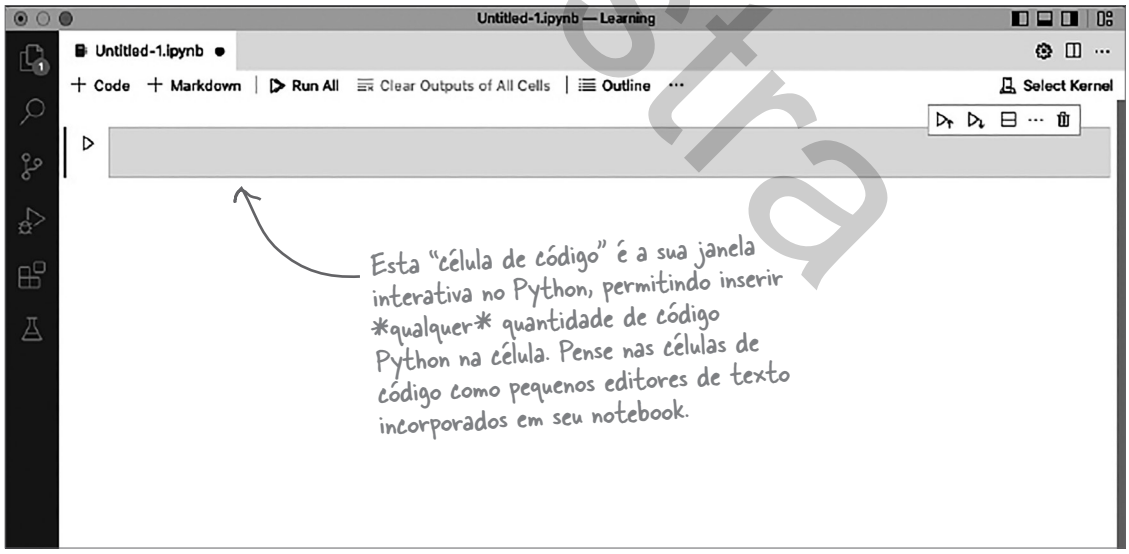
1. Selecione esta opção se você deseja criar um novo arquivo de texto vazio (mas não escolha essa opção agora).

2. Selecione esta opção para criar um arquivo de texto para conter especificamente o código Python (e, novamente, não escolha essa opção no momento).



3. Selecione esta opção para criar um novo "Jupyter Notebook". É a opção que você desejará escolher sempre que iniciar um novo notebook, e — sim! — escolha essa opção agora.

O VS Code cria e abre um novo notebook *untitled* chamado *Untitled-1.ipynb*, que aparece na tela, mais ou menos como:



Esta "célula de código" é a sua janela interativa no Python, permitindo inserir *qualquer* quantidade de código Python na célula. Pense nas células de código como pequenos editores de texto incorporados em seu notebook.

Que rufem os tambores. Agora você está pronto para digitar e rodar algum código Python.

Colocando código no editor do seu notebook

Seu cursor está esperando na célula de código vazia. Antes de digitar qualquer coisa, reserve um momento para rever as primeiras quatro linhas de código do exemplo deste capítulo:

Esta linha importa a tecnologia de randomização do Python para o código do programa.

```
import random

suits = ["Clubs", "Spades", "Hearts", "Diamonds"]
faces = ["Jack", "Queen", "King", "Ace"]
numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Essas três linhas de código criam três variáveis que são atribuídas ao que parece ser um array de strings (os dois primeiros) e um array de números (o último).

Vejamos o que acontece quando você digita esse código no notebook.

Test Drive

Vá em frente e digite as quatro linhas de código mostradas acima na célula em espera. Veja o que temos:

Como cada célula é um pequeno editor integrado, você pode digitar quanto código Python quiser aqui.

```
import random

suits = ["Clubs", "Spades", "Hearts", "Diamonds"]
faces = ["Jack", "Queen", "King", "Ace"]
numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Como a maioria dos editores de código, o VS Code usa o destaque de sintaxe colorido na versão online para tornar seu código mais legível. Esta é a segunda coisa digna de nota.

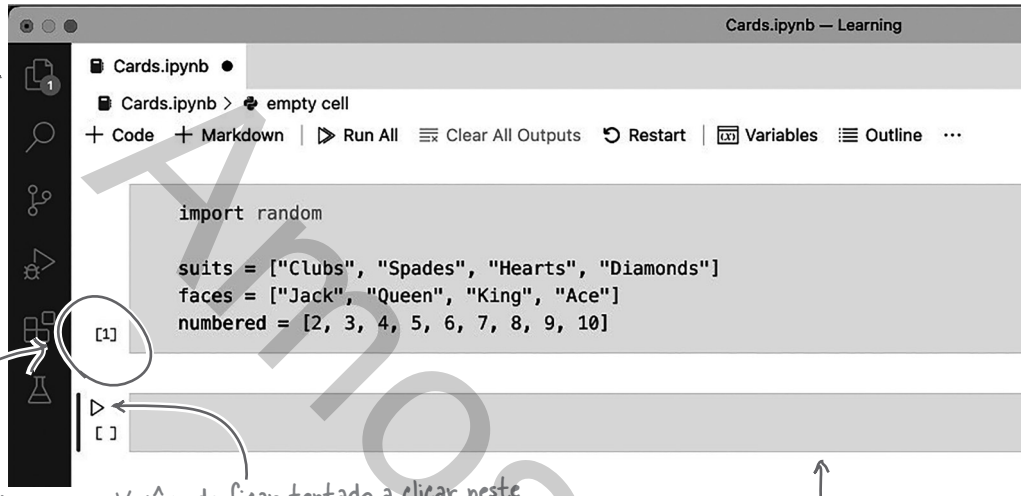
A terceira coisa a notar é que não fica muito claro como executar o código!!!

Shift+Enter para executar o código

Ao pressionar **Shift+Enter**, o código na célula atual selecionada é executado. Depois o *foco* vai para a próxima célula no seu notebook. Se não há uma "próxima célula", o Jupyter cria uma nova:

Quando vir "Shift + Enter" neste livro, pressione e segure a tecla Shift, então, toque na tecla Enter (antes de soltar ambas).

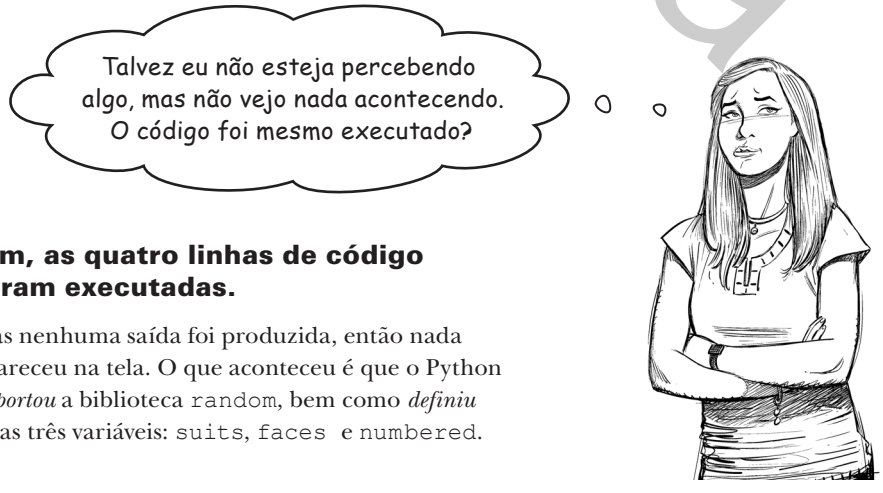
Este pequeno círculo é como o VS Code diz que seu código ainda não foi salvo. Veremos isso em breve.



A célula recebe um identificador numérico exclusivo.

Você pode ficar tentado a clicar neste pequeno ícone de execução, que faz a mesma coisa que "Shift + Enter", mas logo se cansará dessa abordagem porque a combinação de teclas é muito mais rápida.

Como resultado de executar o código na primeira célula, uma nova célula vazia é criada (e o foco vai para ela).



Sim, as quatro linhas de código foram executadas.

Mas nenhuma saída foi produzida, então nada apareceu na tela. O que aconteceu é que o Python *importou* a biblioteca `random`, bem como *definiu* estas três variáveis: `suits`, `faces` e `numbered`.

Exercício

Seu notebook está esperando por mais código. Iremos praticar um pouco com o VS Code adicionando o seguinte código ao seu notebook:

1. Digite `suits` na célula em espera e pressione **Shift+Enter**. Anote aqui o que acontece:

_____

2. Digite o código para a função `draw` na próxima célula e pressione **Shift+Enter** para executar essa célula também. Veja uma cópia do código da função `draw` anterior:

```
def draw():
    the_suit = random.choice(suits)
    the_card = random.choice(faces + numbered)
    return the_card, "of", the_suit
```

Note o uso da indentação para indicar o bloco de código associado à função.

Quando a função é chamada, uma carta aleatória é retornada.

Resposta na página 12

não existem Perguntas Idiotas

P: Estou muito surpreso: as três primeiras linhas de código foram executadas. Com certeza as três variáveis, `suits`, `faces` **e** `numbered`, precisam ser pré-declaradas com algum tipo?

R: Pode ser assim que as outras linguagens de programação funcionam, mas não o Python. As variáveis surgem no momento em que recebem um valor, e esse valor pode ser de qualquer tipo. Não é preciso pré-declarar as informações do tipo, pois o Python trabalha os tipos dinamicamente durante a execução.

P: Importa como eu insiro minha indentação? Posso usar as teclas Tab, Espaço ou as duas?

R: Gostaríamos de dizer que não, mas importa. Quando se trata de indentar seu código Python, é possível usar espaços ou tabulações, mas **não ambos** (nada de misturar as coisas). A razão é bem técnica, então não entraremos em detalhes sobre isso agora. Nosso melhor conselho é configurar seu editor para substituir automaticamente os toques da tecla Tab por quatro espaços. E, sim, há uma *convenção* na comunidade de programação Python para indentar em quatro espaços, não em dois nem oito. Você pode, claro, ignorar a convenção se desejar, mas note que a maioria dos outros programadores Python espera quatro espaços. Claro, há sempre os rebeldes que insistem em fazer do seu próprio jeito.

P: Então, pressionar Enter não executa meu código? Tenho que continuar usando Shift+Enter?

R: Sim, você está operando dentro de um notebook Jupyter. Lembre-se: cada célula em um notebook é como um pequeno editor integrado. Pressionar a tecla **Enter** encerra a linha de código atual e abre uma nova linha (para você inserir mais código). Com a tecla **Enter** já utilizada, o pessoal do Jupyter teve que propor outra forma de executar uma célula de código, estabelecendo **Shift+Enter**.

Aliás: Na última página deste capítulo, você encontrará uma folha de cola útil dos atalhos de teclado Jupyter mais úteis. Por ora, **Shift+Enter** é tudo o que precisa saber.

Exercício Solução

Seu notebook está esperando por mais código e você praticou um pouco com o VS Code adicionando o seguinte código ao seu notebook:

1. Você deveria digitar `suits` na célula em espera, pressionar **Shift+Enter**, então anotar o que acontece:

O valor atual da variável "suits" aparece na tela.

2. Então deveria digitar o código para a função `draw` na próxima célula e pressionar **Shift+Enter** para executar essa célula também. É o que vimos ao concluir o exercício:

O valor atual da variável "suits" aparece na tela quando você só digita o nome da variável em sua célula de código, então pressiona "Shift+Enter".

```
[2]  suits

...  ['Clubs', 'Spades', 'Hearts', 'Diamonds']
```

```
[3]  def draw():
         the_suit = random.choice(suits)
         the_card = random.choice(faces + numbered)
         return the_card, "of", the_suit
```

```
[ ]
```

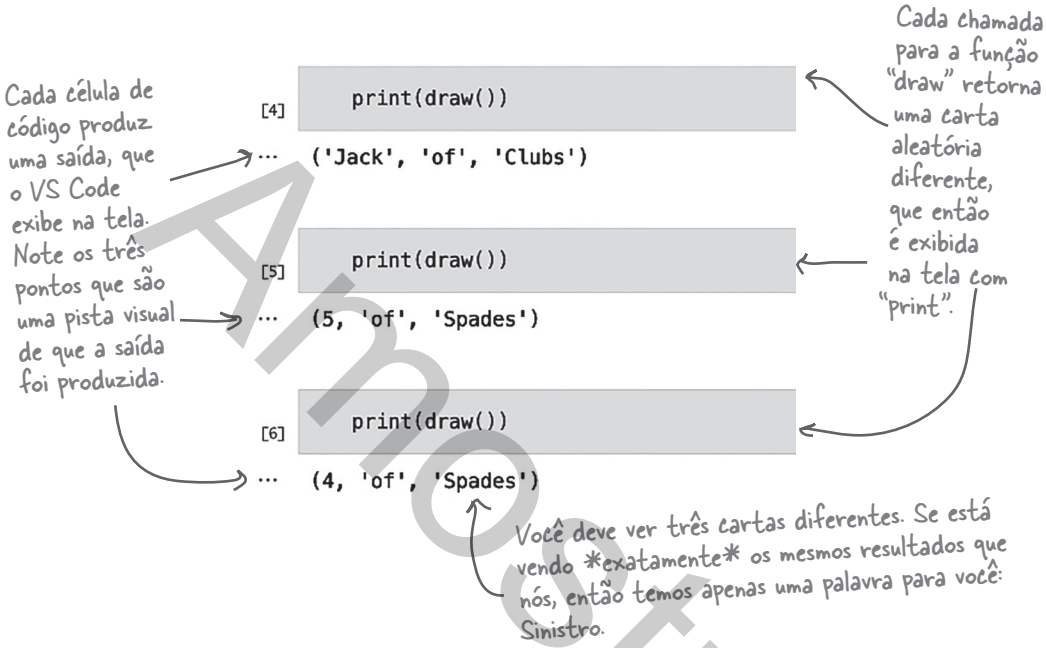
Embora cada célula seja seu próprio pequeno editor integrado, o código nas células seguintes pode se referir às variáveis definidas nas células anteriores. Isso explica por que não tem problema referenciar "suits", "faces" e "numbered" nesta função personalizada, pois tudo está no escopo. Este é um PONTO IMPORTANTE.

Note como o VS Code atribui um número a cada célula executada (com sucesso). É um lembrete visual de que seu código foi executado. As células sem números ainda não foram executadas. Essa funcionalidade é fornecida pelo Jupyter Notebook e não tem nada a ver com o Python, embora muitas vezes seja bom referenciar o número para lembrar em que ordem as células foram executadas (mais sobre isso depois).

Da página 11

Test Drive

Iremos tirar algumas cartas do baralho.
Na próxima célula de código vazia, digite `print(draw()),` então pressione **Shift+Enter**.
Fizemos isso em três células para confirmar que o código está produzindo cartas aleatórias e aqui está o que vimos:



Então... o código Python é realmente fácil de ler... e executar

Além do Jupyter, existem outras formas de executar o código Python, e você aprenderá sobre algumas trabalhando neste livro. No entanto, usar o VS Code com o Jupyter é — em nossa opinião — a maneira *perfeita* de ler, executar, experimentar e *lidar* com o código Python ao aprender pela primeira vez a linguagem. Então prepare-se para passar *muito* tempo no Jupyter e no VS Code.

Antes de continuar, reserve um tempo para selecionar **Arquivo** e depois **Salvar** no menu VS Code para salvar seu notebook com o nome *Cards.ipynb*. ← Faça isto agora!

Sempre preciso usar um notebook Jupyter para rodar meu código Python?

Para saber mais sobre REPL, acesse: https://en.wikipedia.org/wiki/Read–eval–print_loop.

Não, mas é uma ótima ferramenta para usar quando estiver aprendendo a linguagem, já que o Jupyter se conecta ao *REPL* predefinido do Python para executar cada uma das células de código. Você aprenderá a executar seu código *fora* de um notebook mais adiante neste livro, mas, por ora, só precisa se preocupar em usar os notebooks dentro do VS Code (conforme acompanha).

Diz aqui que eu não preciso compilar e vincular meu código Python a um executável. É verdade?

Sim. O Python é um **interpretador** planejado para executar cada linha de código como e quando ele a vê. Para ser tecnicamente correto, o Python *compila* seu código, mas tudo acontece nos bastidores (para aliviar a carga de fazer isso sozinho). Você só precisa escrever e executar seu código: o Python cuida dos detalhes e o mecanismo **Shift+Enter** do Jupyter facilita que você controle quando a execução acontece.

Para esclarecer, tudo o que vejo em uma caixa cinza neste livro é uma simulação de uma célula de código?

Sim, quando você vê um código apresentado dentro de uma caixa cinza, pode presumir que é o código a ser inserido em uma célula de código no seu notebook atual pronto para a execução com **Shift+Enter**. Para acompanhar, digite o código em cada caixa cinza no seu notebook e execute-o (quando necessário) com **Shift+Enter**.

Ótimo. Essas respostas ajudam muito. Agora, voltando ao exemplo da carta... Se eu quisesse tirar cinco cartas, usaria um loop em vez de chamar "draw" cinco vezes, certo?

Certo! E o loop **for** do Python é o que você usaria. Vejamos rapidamente como usar **for** para isso.

E se você quiser mais de uma carta?

Sua função `draw` é um ótimo começo, tirando uma carta do baralho cada vez que a função é executada. Mas, e se quiser tirar mais de uma carta?

Embora seja um pouco ridículo sugerir chamar manualmente sua função `draw` quantas vezes forem necessárias, a maioria dos programadores recorre a um loop. Você aprenderá mais sobre os loops do Python mais adiante neste livro. Por enquanto, veja como usaria o loop **for** do Python para executar a função `draw` cinco vezes:

A palavra-chave "for" inicia o loop.

"range" em outra função predefinida (BIF) que, neste caso, está sendo usada para controlar quantas vezes o código do loop é executado.

```
for _ in range(5):
    print(draw())
```

O código do loop é indentado "sob" a palavra-chave "for".

O bloco de código do loop "for" é uma única linha de código, mas você pode ter quantas linhas precisar aqui.

Não me diga que só eu notei isso, mas... qual é a desse caractere underscore?

É a variável padrão do Python.

É normal que a maioria dos loops tenha uma variável de loop associada (com "`i`" sendo o nome favorito em particular para muitos programadores). Porém, se o código do loop não usa o valor dessa variável, o Python permite usar o caractere underscore.

Quando você vir o underscore no código, pense: "*Ah ha! Uma variável é necessária sintaticamente aqui, mas seu valor não é usado, então a variável não foi nomeada*".

Test Drive

Iremos testar seu primeiro loop em Python e veremos o que acontece. Estamos mostrando o código em uma caixa cinza, então digite o código na próxima célula de código do notebook e pressione **Shift+Enter**:

O código do loop da última página (mostrado em uma célula de código).

```
for _ in range(5):
    print(draw())
```

Só para sua informação: a partir de agora, quando exibirmos uma célula de código, não mostraremos os números das células Jupyter, pois queremos que você se concentre no código.

E, como esperado, os detalhes das cinco cartas aparecem na tela:

```
(5, 'of', 'Clubs')
('Queen', 'of', 'Diamonds')
('Queen', 'of', 'Hearts')
('King', 'of', 'Diamonds')
('Queen', 'of', 'Diamonds')
```

Parece bom... ou não? Tem algo errado bem aqui?

Percebi na hora. Imagino que haja um problema ao tirar a mesma carta duas vezes, certo? Seu baralho não pode ter duas Damas de Ouros...

Sim, a mesma carta sendo tirada duas vezes.

O problema aqui é que a função para tirar cartas está retornando a *Dama de Ouros* duas vezes. Ops!

Vejamos melhor o que `draw` está fazendo para saber por que isso acontece.

Veja melhor o código para tirar cartas

A função `draw` tem apenas três linhas de comprimento, mas ainda impressiona. Veja:

1. A primeira linha de código no corpo da função seleciona *aleatoriamente* um naipe, atribuindo o valor a uma variável devidamente nomeada.

```
def draw():
    the_suit = random.choice(suits)
    the_card = random.choice(faces + numbered)
    return the_card, "of", the_suit
```

2. A segunda linha de código combina as figuras e as cartas numeradas, depois seleciona *aleatoriamente* um valor, mais uma vez atribuindo o valor a uma variável devidamente nomeada.

3. Com as duas variáveis atribuídas *aleatoriamente*, a terceira e última linha de código retorna a carta selecionada.

Sabemos: todos aqueles caracteres * na palavra "aleatoriamente" nessas anotações não eram pistas suficientes, eram? Embora o código `draw` tenha sucesso ao selecionar uma carta aleatória, ele não evita o retorno do mesma carta duas vezes. Afinal, na maioria dos jogos de cartas, quando uma carta é selecionada no baralho, ela raramente é colocada de volta, certo? Se este fosse um código "real" para tirar cartas (não um exemplo inventado e planejado para dar suporte à visão geral do Python deste capítulo), provavelmente usaríamos uma estrutura de dados diferente dos arrays utilizados atualmente, concorda?

Poder do Cérebro

Se fosse pedido para você criar uma estrutura de dados que modelasse melhor um baralho de cartas, qual estrutura escolheria?

Os 4 Principais: lista, tupla, dicionário e conjunto

É lendário o excelente suporte predefinido do Python para as estrutura de dados e é normalmente citado como a principal razão para a maioria dos programadores Python *amar* o Python.

Como este é o seu capítulo de abertura, não iremos sobrecarregá-lo com qualquer discussão aprofundada dessas estruturas de dados agora. Há muitas páginas (capítulos inteiros, na verdade) dedicadas aos *4 Principais* mais adiante neste livro.

Embora não tenhamos mencionado especificamente seu uso, você já viu listas *e* tuplas. Apesar de nos referirmos a elas como *arrays* antes, cada um desses "arrays" é, na verdade, uma **lista** Python autêntica de fato:

As listas começam e terminam com colchete.

```
suits = ["Clubs", "Spades", "Hearts", "Diamonds"]
faces = ["Jack", "Queen", "King", "Ace"]
numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Cada item em uma lista é separado por vírgula.

Você também já teve sua cota justa de tuplas. Sempre que você chama a função `draw`, ela retorna uma **tupla**:

Uma pergunta rápida: como você pronunciaria "tupla"? Rimando com "dupla" ou com "quádrupla"? É um mistério. 😉

```
draw()
(3, 'of', 'Clubs')
```

É uma tupla (e parece uma lista).

Ao contrário das listas, que têm colchetes, as tuplas têm parênteses em sua coleção de valores.

Ninguém o culparia por achar as tuplas meio esquisitas e teríamos que concordar que as consideramos esquisitas também. Não se preocupe com isso.

Você aprenderá mais sobre listas e tuplas mais adiante no livro. Embora ambas tenham seus usos, não são uma ótima opção quando se trata de modelar um baralho de cartas. Uma outra estrutura de dados é necessária aqui. Mas, qual?

Dica: há uma ótima pista no título desta página.

Modele o baralho de cartas com um conjunto

Os conjuntos em Python são como os conjuntos da aula de matemática: eles contêm uma coleção de valores exclusivos em que duplicatas *não* são permitidas.

Os conjuntos em Python também são especialmente convenientes para adicionar e remover objetos. E, sim, outras estruturas de dados podem adicionar/remover objetos também, mas — normalmente — você tem que encontrar o objeto primeiro, depois removê-lo. Ou tem que pesquisar o objeto e assegurar que ele já não existe na estrutura de dados *antes* de adicioná-lo. Nos dois casos, são duas operações para adicionar e duas para remover (com outras estruturas de dados).

Não é assim com os conjuntos.

Uma única operação adiciona ao conjunto e outra remove do conjunto, evitando a necessidade de fazer toda essa pesquisa. Na nossa cabeça, usar um conjunto para modelar um baralho de cartas é uma combinação quase perfeita.

Continue **acompanhando** em seu notebook *Cards.ipynb* enquanto criamos primeiro um conjunto vazio, então aprenda um pouco sobre conjuntos antes de adicionar as 52 cartas e removê-las do baralho quando necessário.

Não se preocupe se pensou em "dicionário" (afinal, sempre será uma coisa ou outra). Você aprenderá tudo sobre dicionários mais tarde neste livro.

Criar um novo conjunto vazio é fácil. Chame a função predefinida (BIF) "set", então atribua o conjunto a uma variável.

```
deck = set()
```

Não se esqueça de usar "Shift+Enter" para executar o código nas células de código.

Agora que seu conjunto existe e está atribuído a uma variável (com um nome bem criativo) chamada `deck`, aprenderemos um pouco sobre isso. Duas das funções predefinidas (BIF) do Python ajudam aqui: `type` e `len`:

"BIF" é uma abreviação de "função predefinida".

A BIF "type" confirma que sua variável "deck" se refere a um conjunto.

```
type(deck)
```

```
set
```

A BIF "len" confirma que seu conjunto "deck" está vazio no momento.

```
len(deck)
```

```
0
```

Agora que seu conjunto existe, como determinar o que você pode fazer com ele? Bem... há uma BIF para isso.

O combo mambo print dir

Quando fornecido o nome de qualquer objeto Python, a BIF **dir** retorna uma lista dos atributos do objeto que, no caso da variável `deck`, são os atributos associados a um objeto do conjunto.

Como você pode ver (abaixo), há muitos atributos associados a um conjunto Python. Observe como a saída de **dir** é combinada com uma chamada para a BIF **print**, assegurando que a saída exibida seja mostrada na *horizontal* da tela, não na *vertical*, o que reduz a quantidade de rolagem necessária para seus pobres dedos. Pode não ser grande coisa, mas — ei! — qualquer ajuda é bem-vinda.

Eles estão dançando o "combo mambo". Mas... qual é "print" e qual é "dir"?!?

Alguns desses nomes de atributos parecem muito estranhos, em especial aqueles com caracteres de underscore duplo antes e depois.

```
print(dir(deck))
```

```
['__and__', '__class__', '__class_getitem__', '__contains__',
'__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getstate__', '__gt__', '__hash__', '__iand__',
'__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__',
'__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__',
'__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__',
'__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference',
'difference_update', 'discard', 'intersection', 'intersection_update',
'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```

O resto dos nomes é mais fácil de ler, mas — Céus! — há muitos, não é?

Se a saída for para a direita da tela (ao invés de para baixo), veja se você selecionou a opção "Notebook > Saída: Quebra de Linha". do VS Code. Você pode pesquisar essa configuração digitando "quebra de linha" na barra de pesquisa Configurações do VS Code.

Veja uma regra simples para seguir ao ver a saída de **print dir**: *Por ora, ignore os atributos que começam e terminam com um underscore duplo.* Você aprenderá por que eles existem mais tarde neste livro, mas — no momento — ignore, ignore, ignore!

Tendo ajuda com a saída de dir

Pode não parecer à primeira vista, mas provavelmente você usará a BIF **dir** mais do que qualquer outra ao trabalhar com o Python, sobretudo ao experimentar em um notebook Jupyter. Isso é devido à capacidade de **dir** de divulgar a lista de atributos associados a qualquer objeto. Em geral, esses atributos incluem uma lista de *métodos* que podem ser aplicados ao objeto.

Embora seja tentador (ainda que um pouco absurdo) executar aleatoriamente qualquer método associado à variável `deck` para ver o que ele faz, uma abordagem mais sensata é ler a documentação referente a qualquer método…

Mas não se preocupe: não pediremos que percorra milhares de páginas de documentação online do Python. Isso é trabalho da BIF **help**:

Em uma célula de código vazia, use a BIF "help" para exibir a documentação de qualquer método do objeto.

```
help(deck.add)
```

```
Help on built-in function add:

add(...) method of builtins.set instance
    Add an element to a set.

    This has no effect if the element is already present.
```

Esta linha de saída é a parte importante aqui.

Também é possível ver a documentação dentro do VS Code passando o mouse sobre a palavra "add" para ver uma dica de ferramenta. Mas gostamos de usar a BIF "help", pois os docs ficam na tela e não desaparecem no momento em que movemos o mouse (removendo a dica).

Preencha o conjunto com cartas

No final da última página (e por sorte), usamos a BIF **help** para ver a documentação do Python para o método **add**, que é criado em cada conjunto Python. Sabendo o que **add** faz, vejamos um código que usa os dados de carta "brutos" de antes para criar um baralho:

Veja os dados de carta "brutos" (em três listas).

```
suits = ["Clubs", "Spades", "Hearts", "Diamonds"]
faces = ["Jack", "Queen", "King", "Ace"]
numbered = [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Usamos um loop dentro de um loop para fazer o trabalho pesado. O loop externo percorre os naipes, um de cada vez: Paus, Espadas, Copas e Ouros.

O loop interno combina todos os possíveis valores das cartas em uma lista, depois percorre cada um deles, adicionando as 52 cartas ao baralho.

```
for suit in suits:
    for card in faces + numbered:
        deck.add((card, "of", suit))
```

O método "add" é chamado em cada carta para aumentar o conjunto.

Ao contrário do código de loop anterior, que usou a variável padrão do Python (o underscore), nestes loops nomeamos cada variável de loop como "suit" e "card", respectivamente.

```
len(deck)
```

```
52
```

Com o código de loop executado, a BIF "len" é usada para confirmar que nosso conjunto não está mais vazio.

Lembrou-se de acompanhar e usar "Shift+Enter" para executar as células?

Sim. E tudo bem.

As estruturas de dados do Pyhon podem aumentar e diminuir quando necessário, por isso não é preciso pré-declarar seu tamanho.

O interpretador Python lida com todos os detalhes subjacentes para você. A memória é alocada e desalocada dinamicamente quando necessário, liberando você para criar o código de que precisa.

Parece um baralho de cartas agora

Agora que seu baralho é um conjunto, você pode modelar melhor o comportamento.

Infelizmente, selecionar aleatoriamente uma carta do baralho é complicado pelo fato de que a técnica `random.choice` anterior no capítulo não funciona com os conjuntos . É uma pena, pois seria bom usar `random.choice(deck)` para escolher uma carta, mas — enfim — não funcionará.

Por ora, não se preocupe com o motivo.

Não se preocupe. Uma alteração rápida permite que você primeiro *converta* uma cópia do conjunto de cartas em uma lista, que pode então ser usada com `random.choice`. Não poderia ser mais simples:

Quando você vir um código como este, leia de dentro para fora. Pegue o conjunto de cartas na variável "deck" e converta-o em uma lista. Depois, alimente a lista à função "random.choice".

Atribua a carta selecionada aleatoriamente a uma nova variável chamada "card".

```
card = random.choice(list(deck))
print(card)
```

```
('Ace', 'of', 'Spades')
```

Uma carta aleatória é mostrada, embora os fãs da banda Motorhead possam discordar. 😉

Exiba o valor de "card" na tela.

Tendo selecionado uma carta (o *Ás de Espadas* para nós, mas provavelmente é uma carta diferente se você vem acompanhando), devemos realmente retirar a carta do baralho para que as escolhas aleatórias subsequentes não a selecionem de novo.

A função "remove", predefinida em todos os conjuntos Python, é o que você precisa aqui.

```
deck.remove(card)
len(deck)
```

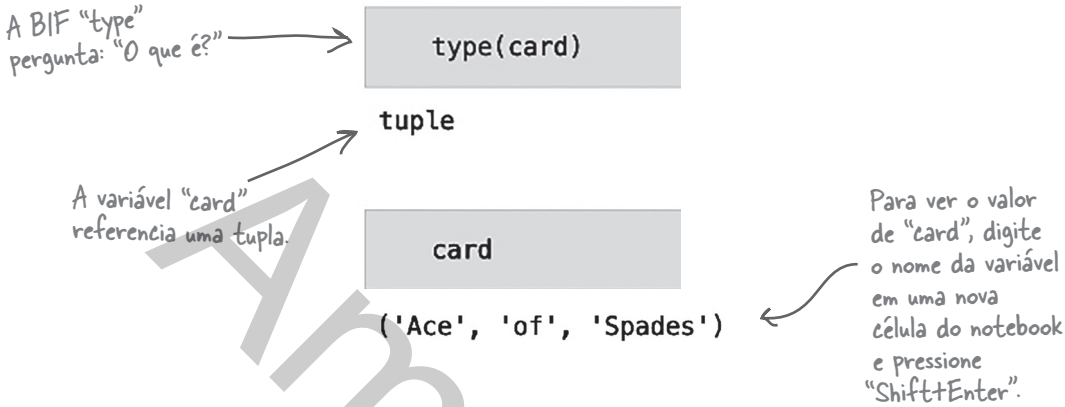
```
51
```

Como seria de esperar, o baralho diminui de tamanho após cada "remove" bem-sucedido.

Se quiser saber um pouco mais sobre "remove", lembre-se de que pode digitar "help(deck.remove)" em uma célula de código vazia para ver seus docs.

O que é exatamente "card"?

Se você está se perguntando o que é a variável `card`, não se esqueça de **type**, que informa o tipo do valor atribuído atualmente a `card`:



Aponte seu lápis

Agora que você já viu como selecionar aleatoriamente uma carta no conjunto, bem como remover uma carta, iremos reescrever a função `draw` para trabalhar com o conjunto `deck` (não com as listas). No espaço abaixo, forneça as três linhas de código que você adicionaria à sua nova função `draw`. A primeira linha de código seleciona aleatoriamente uma carta no conjunto, a segunda remove a carta selecionada do conjunto e a terceira (e última) linha de código retorna a carta selecionada para o chamador da função:

```
def draw():
    _____
    _____
    _____
    _____
    _____
```

Adicione suas três linhas de código aqui (e não se esqueça de indentar).

Resposta na página 26