Python para Excel

Um Ambiente Moderno para Automação e Análise de Dados

Felix Zumstein

ALTA BOOKS
GRUPO EDITORIAL
Rio de Janeiro, 2024

Sumário

PARTE II

Introdução ao pandas

PARTE IV

Programando o Aplicativo Excel com xlwings

CAPÍTULO 1

Por que Python para Excel?

Normalmente, os usuários do Excel começam a questionar suas ferramentas de planilha quando atingem uma limitação. Um exemplo clássico é quando as pastas de trabalho do Excel contêm tantos dados e fórmulas que se tornam lentas ou, na pior das hipóteses, travam. No entanto, faz sentido questionar sua configuração antes que as coisas deem errado: se você trabalha em pastas de trabalho de importância vital, nas quais os erros podem resultar em danos financeiros ou de reputação, ou se você gasta horas todos os dias atualizando manualmente as pastas de trabalho do Excel, deve aprender a automatizar seus processos com uma linguagem de programação. A automação elimina o risco de erro humano e permite que você gaste seu tempo em tarefas mais produtivas do que copiar/colar dados em uma planilha do Excel.

Neste capítulo, apresentarei algumas razões pelas quais o Python é uma excelente escolha em combinação com o Excel e quais vantagens são comparadas à linguagem de automação integrada do Excel, o VBA. Depois de apresentar o Excel como linguagem de programação e entender suas particularidades, apontarei os recursos específicos que tornam o Python muito mais forte em comparação com o VBA. Para começar, no entanto, daremos uma olhada rápida nas origens de nossos dois personagens principais!

Em termos de tecnologia computacional, o Excel e o Python existem há muito tempo: o Excel foi lançado pela primeira vez em 1985 pela Microsoft — e isso pode ser uma surpresa para muitos —, estando disponível apenas para Apple Macintosh. Não foi até 1987 que o Microsoft Windows obteve sua primeira versão na forma do Excel 2.0. A Microsoft não foi o primeiro player no mercado de planilhas eletrônicas: a VisiCorp saiu com o VisiCalc em 1979, seguido pela Lotus Software em 1983 com o Lotus 1-2-3. E a Microsoft não liderou com o Excel: três anos antes, eles lançaram o Multiplan, um programa de planilhas que poderia ser usado no MS-DOS e em alguns outros sistemas operacionais, mas não no Windows.

O Python nasceu em 1991, apenas seis anos depois do Excel. Embora o Excel tenha se tornado popular desde o início, o Python demorou um pouco mais até ser adotado em certas áreas, como desenvolvimento web ou administração de

sistemas. Em 2005, o Python começou a se tornar uma alternativa séria para a computação científica quando o *NumPy*, um pacote para computação baseada em array e álgebra linear, foi lançado pela primeira vez. O NumPy combinou dois pacotes predecessores e, portanto, simplificou todos os esforços de desenvolvimento em torno da computação científica em um único projeto. Hoje, ele forma a base de inúmeros pacotes científicos, incluindo o *pandas*, lançado em 2008 e que é o grande responsável pela ampla adoção do Python no mundo da ciência de dados e finanças que começou a acontecer depois de 2010. Graças ao pandas, o Python, juntamente com o R, tornou-se uma das linguagens mais usadas para tarefas de ciência de dados, como análise de dados, estatísticas e aprendizado de máquina.

O fato de o Python e o Excel terem sido inventados há muito tempo não é a única coisa que eles têm em comum: também são uma linguagem de programação. Embora você provavelmente não fique surpreso ao ouvir isso sobre o Python, pode necessitar de uma explicação para o Excel, que darei a seguir.

O Excel É uma Linguagem de Programação

Esta seção começa apresentando o Excel como uma linguagem de programação, que o ajudará a entender por que os problemas de planilhas aparecem regularmente nas notícias. Em seguida, veremos algumas práticas recomendadas que surgiram na comunidade de desenvolvimento de software e podem evitar muitos erros típicos do Excel. Concluiremos com uma breve introdução ao Power Query e ao Power Pivot, duas ferramentas modernas do Excel que cobrem o tipo de funcionalidade para a qual usaremos o pandas.

Se você usa o Excel para mais do que sua lista de compras, definitivamente está usando funções como `=SOMA(A1:A4)` para somar um intervalo de células. Se pensar um pouco sobre como isso funciona, notará que o valor de uma célula geralmente depende de uma ou mais células, que podem novamente usar funções que dependem de uma ou mais células, e assim por diante. Fazer essas chamadas de função aninhadas não é diferente de como outras linguagens de programação funcionam, você apenas escreve o código em células em vez de arquivos de texto. E se isso ainda não o convenceu: no fim de 2020, a Microsoft anunciou a introdução das *funções lambda*, que permitem escrever funções reutilizáveis na própria linguagem de fórmulas do Excel, ou seja, sem precisar depender de uma linguagem diferente, como o VBA. De acordo com Brian Jones, chefe de produto do Excel, essa era a peça que faltava para finalmente tornar o Excel uma linguagem de programação "real".[1] Isso também significa que os usuários do Excel deveriam realmente ser chamados de programadores do Excel!

1 Você pode ler o anúncio das funções lambda no blog do Excel.

Há algo especial, porém, sobre os programadores do Excel: a maioria deles são usuários de negócios ou especialistas de domínio sem educação formal em ciência da computação. São comerciantes, contadores ou engenheiros, para citar apenas alguns exemplos. Suas ferramentas de planilhas são projetadas para resolver um problema de negócios e muitas vezes ignoram as melhores práticas no desenvolvimento de software. Como consequência, essas ferramentas muitas vezes misturam entradas, cálculos e saídas nas mesmas planilhas, podem exigir a execução de etapas não óbvias para que funcionem corretamente e alterações críticas são feitas sem qualquer rede de segurança. Em outras palavras, as ferramentas de planilhas carecem de uma arquitetura de aplicativo sólida e geralmente não são documentadas nem testadas. Às vezes, esses problemas podem ter consequências devastadoras: se você esquecer de recalcular sua planilha de trading antes de fazer uma negociação, poderá comprar ou vender o número errado de ações, o que pode fazer com que perca dinheiro. E, se não é apenas seu próprio dinheiro que você está negociando, poderemos ler sobre isso nas notícias, como veremos a seguir.

Excel no Noticiário

O Excel é um convidado regular nas notícias e, durante a redação deste artigo, duas novas histórias chegaram às manchetes. A primeira foi sobre o HUGO Gene Nomenclature Committee, que renomeou alguns genes humanos para que não fossem mais interpretados pelo Excel como datas. Por exemplo, para evitar que o gene `MARCH1` fosse transformado em `1-Mar`, ele foi renomeado como `MARCHF1`.[2] Na segunda história, o Excel foi responsabilizado pelo atraso na comunicação de 16 mil resultados de testes de Covid-19 na Inglaterra. O problema foi causado porque os resultados do teste foram gravados no formato de arquivo do Excel mais antigo (*.xls*), limitado a aproximadamente 65 mil linhas. Isso significava que conjuntos de dados maiores foram simplesmente cortados além desse limite.[3] Embora essas duas histórias mostrem a importância e o domínio contínuos do Excel no mundo de hoje, provavelmente não há outro "incidente de Excel" mais famoso do que o London Whale [*Baleia de Londres*, em tradução livre].

London Whale é o apelido de um trader cujos erros de negociação forçaram o JP Morgan a anunciar uma perda impressionante de US$6 bilhões em 2012. A fonte da explosão foi um modelo de valor em risco baseado em Excel que estava subestimando substancialmente o verdadeiro risco de perder dinheiro em uma de suas carteiras. O *Relatório do JPMorgan Chase & Co. Management Task Force*

2 James Vincent, "Scientists rename human genes to stop Microsoft Excel from misreading them as dates", *The Verge*, 6 de agosto de 2020, *https://oreil.ly/0qo-n* (conteúdo em inglês).

3 Leo Kelion, "Excel: Why using Microsoft's tool caused COVID-19 results to be lost", *BBC News*, 5 de outubro de 2020, *https://oreil.ly/vvB6o* (conteúdo em inglês).

Regarding 2012 CIO Losses[4] (2013) menciona que "o modelo operava por meio de uma série de planilhas Excel, que precisavam ser preenchidas manualmente, por um processo de copiar e colar dados de uma planilha para outra". Além desses problemas operacionais, elas tinham um erro lógico: em um cálculo, elas estavam dividindo por uma soma, em vez de por uma média.

Se você quiser ver mais dessas histórias, dê uma olhada em Horror Stories, uma página da web mantida pelo European Spreadsheet Risks Interest Group (EuSpRIG).

Para evitar que sua empresa acabe virando notícia com uma história semelhante, daremos uma olhada em algumas das melhores práticas recomendadas a seguir que tornam seu trabalho com o Excel muito mais seguro.

Melhores Práticas de Programação

Esta seção apresentará as melhores práticas de programação mais importantes, incluindo a separação de conceitos, o princípio DRY, testes e controle de versão. Como veremos, segui-las será mais fácil quando você começar a usar o Python junto do Excel.

Separação de conceitos

Um dos princípios de design mais importantes na programação é a *separação de conceitos*, às vezes também chamada de *modularidade*. Isso significa que um conjunto relacionado de funcionalidades deve ser cuidado por uma parte independente do programa para que possa ser facilmente substituído sem afetar o restante do aplicativo. No nível mais alto, um aplicativo geralmente é dividido nas seguintes camadas:[5]

- Camada de apresentação
- Camada de negócios
- Camada de dados

Para explicar essas camadas, considere um conversor de moeda simples como o mostrado na Figura 1-1. Você encontrará o arquivo do Excel *currency_converter.xlsx* na pasta *xl* do material complementar.

É assim que o aplicativo funciona: digite o Valor e a Moeda nas células A4 e B4, respectivamente, e o Excel converterá em dólares norte-americanos na célula D4. Muitos aplicativos de planilha seguem esse design e são utilizados pelas empresas todos os dias. Deixe-me dividir o aplicativo em suas camadas:

4 A Wikipédia tem o link para o documento em uma das notas de rodapé em seu artigo sobre o caso. Disponível em https://en.wikipedia.org/wiki/2012_JPMorgan_Chase_trading_loss#cite_note-35 (conteúdo em inglês).

5 A terminologia é retirada do *Guia de Arquitetura de Aplicativos da Microsoft, 2ª edição*. Disponível em https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff650706(v=pandp.10) (conteúdo em inglês).

Camada de apresentação

É o que você vê e interage, ou seja, a interface do usuário: os valores das células A4, B4 e D4 junto aos seus rótulos constroem a camada de apresentação do conversor de moeda.

Camada de negócios

Essa camada cuida da lógica específica do aplicativo: a célula D4 define como o valor é convertido em USD. A fórmula `=A4 * PROCV(B4, F4:G11, 2, FALSE)` se traduz em Valor vezes Taxa de câmbio.

Camada de dados

Como o nome sugere, essa camada cuida do acesso aos dados: a parte `PROCV` da célula D4 faz esse trabalho

A camada de dados acessa os dados da tabela de câmbio que inicia na célula F3 e funciona como o banco de dados desse pequeno aplicativo. Se você prestou bastante atenção, provavelmente notou que a célula D4 aparece em todas as três camadas: esse aplicativo simples mistura as camadas de apresentação, de negócios e de dados em uma única célula.

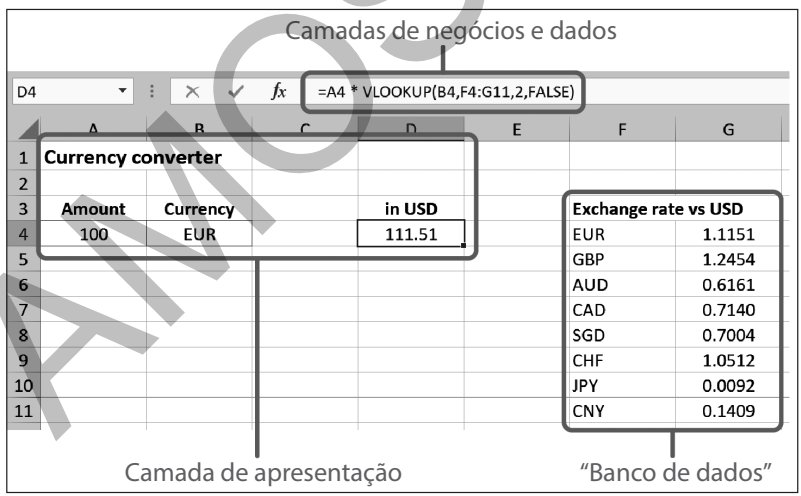

Figura 1-1. currency_converter.xlsx

Não é necessariamente um problema para esse conversor de moeda simples, mas, muitas vezes, o que começa como um pequeno arquivo do Excel se transforma em um aplicativo muito maior. Como essa situação pode ser melhorada? A maioria dos recursos profissionais para desenvolvedores do Excel aconselha você a usar uma planilha separada para cada camada, na terminologia do Excel geralmente

chamada de *entradas*, *cálculos* e *saídas*. Muitas vezes, isso é combinado com a definição de um determinado código de cor para cada camada, por exemplo, um fundo azul para todas as células de entrada. No Capítulo 11, construiremos um aplicativo real baseado nessas camadas: o Excel será a camada de apresentação, enquanto as camadas de negócios e de dados serão movidas para o Python, onde é muito mais fácil estruturar seu código adequadamente.

Agora que você sabe o que significa a separação de conceitos, descobriremos o que é o princípio DRY!

Princípio DRY

O livro *O Programador Pragmático,* de Hunt e Thomas (Bookman), popularizou o princípio DRY: *don't repeat yourself* [não se repita, em tradução livre]. Nenhum código duplicado significa menos linhas de código e menos erros, o que facilita a manutenção do código. Se a sua lógica de negócios estiver nas fórmulas da sua célula, é praticamente impossível aplicar o princípio DRY, pois não haverá um mecanismo que permita reutilizá-lo em outra pasta de trabalho. Isso, infelizmente, significa que uma maneira comum de iniciar um novo projeto do Excel é copiar a pasta de trabalho do projeto anterior ou de um modelo.

Se você escreve VBA, a parte mais comum do código reutilizável é uma função. Uma função dá acesso ao mesmo bloco de código de várias macros, por exemplo. Se você tiver várias funções que usa o tempo todo, talvez queira compartilhá-las entre as pastas de trabalho. O instrumento padrão para compartilhar o código VBA entre as pastas de trabalho são os suplementos, mas os suplementos VBA não possuem uma maneira robusta de distribuí-los e atualizá-los. Embora a Microsoft tenha introduzido um armazenamento de suplementos internos do Excel para resolver esse problema, isso só funciona com suplementos baseados em JavaScript, portanto não é uma opção para os codificadores VBA. Isso significa que ainda é muito comum usar a abordagem copiar/colar com VBA: iremos supor que você precise de uma função *spline cúbica* no Excel. Essa função é uma maneira de interpolar uma curva com base em alguns pontos em um sistema de coordenadas e é frequentemente usada por operadores de renda fixa para derivar uma curva da taxa de juros para todos os vencimentos com base em algumas combinações conhecidas de vencimento/taxa de juros. Se você pesquisar por "Spline Cúbica Excel" na internet, não demorará muito até ter uma página de código VBA que faça o que deseja. O problema com isso é que, muito comumente, essas funções foram escritas por uma única pessoa, provavelmente com boas intenções, mas sem documentação formal ou teste. Talvez elas funcionem para a maioria das entradas, mas e os casos extremos? Se você está negociando uma carteira de renda fixa multimilionária, quer ter algo que sabe que pode confiar.

Pelo menos, é isso que você ouvirá de seus auditores internos quando descobrirem de onde vem o código.

O Python facilita a distribuição de código usando um gerenciador de pacotes, como veremos na última seção deste capítulo. Antes de chegarmos lá, no entanto, continuaremos com os testes, um dos pilares do desenvolvimento de software sólido.

Testes

Quando você diz a um desenvolvedor do Excel para testar suas pastas de trabalho, ele provavelmente realizará algumas verificações aleatórias: clicará em um botão e verá se a macro ainda faz o que deveria fazer ou alterará algumas entradas e verificará se a saída parece razoável. No entanto, é uma estratégia arriscada: o Excel facilita a introdução de erros difíceis de detectar. Por exemplo, você pode substituir uma fórmula por um valor codificado diretamente. Ou se esquecer de ajustar uma fórmula em uma coluna oculta.

Quando você diz a um desenvolvedor de software profissional para testar seu código, ele escreve *testes de unidade*. Como o nome sugere, é um mecanismo para testar componentes individuais do programa. Por exemplo, os testes de unidade garantem que uma única função de um programa opere corretamente. A maioria das linguagens de programação oferece uma maneira de executar testes de unidade automaticamente. A execução de testes automatizados aumentará drasticamente a confiabilidade de sua base de código e dará uma segurança razoável de que você não violará nada que funciona atualmente ao editar seu código.

Se você observar a ferramenta de conversão de moeda na Figura 1-1, poderá escrever um teste que verifica se a fórmula na célula D4 retorna corretamente USD 105 com as seguintes entradas: 100EUR como valor e 1,05 como taxa de câmbio EURUSD. Por que isso ajuda? Suponha que você exclua acidentalmente a célula D4 com a fórmula de conversão e precise reescrevê-la: em vez de multiplicar o valor pela taxa de câmbio, você divide por ela — afinal, trabalhar com moedas pode ser confuso. Ao executar o teste acima, você obterá uma falha no teste, pois 100EUR/1,05 não resultará mais em 105USD, como o teste espera. Assim, você pode detectar e corrigir a fórmula antes de entregar a planilha aos seus usuários.

Praticamente todas as linguagens de programação tradicionais oferecem uma ou mais estruturas de teste para escrever testes de unidade sem muito esforço — mas não o Excel. Felizmente, o conceito de testes de unidade é bastante simples e, ao conectar o Excel com o Python, você obtém acesso às poderosas estruturas de teste unitário do Python. Embora uma apresentação mais aprofundada dos tes-

tes de unidade esteja além do escopo deste livro, convido você a dar uma olhada na postagem em meu blog,[6] na qual eu apresento o tópico com exemplos práticos.

Os testes de unidade geralmente são configurados para serem executados automaticamente quando você envia seu código para o sistema de controle de versão. A próxima seção explica o que são sistemas de controle de versão e por que eles são difíceis de usar com arquivos do Excel.

Controle de versão

Outra característica dos programadores profissionais é que eles utilizam um sistema de *controle de versão* ou *controle de fonte*. Um *sistema de controle de versão* [em inglês, *version control system* — VCS] rastreia as alterações em seu código-fonte ao longo do tempo, permitindo que você veja quem alterou o quê, quando e por que, e permite reverter para versões antigas a qualquer momento. O sistema de controle de versão mais popular hoje é o Git. Ele foi originalmente criado para gerenciar o código-fonte do Linux e, desde então, conquistou o mundo da programação — até a Microsoft adotou o Git em 2017 para gerenciar o código-fonte do Windows. No mundo do Excel, por outro lado, o sistema de controle de versão de longe mais popular vem na forma de uma pasta na qual os arquivos são arquivados assim:

```
currency_converter_v1.xlsx
currency_converter_v2_2020_04_21.xlsx
currency_converter_final_edits_Bob.xlsx
currency_converter_final_final.xlsx
```

Se, ao contrário desse exemplo, o desenvolvedor do Excel segue determinada convenção no nome do arquivo, não há nada errado com isso. Mas manter um histórico de versão de seus arquivos localmente impede você de ter acesso a aspectos importantes do controle de fonte com uma colaboração mais fácil, revisões dos colegas, processos de aprovação e logs de auditoria. E, se você deseja tornar suas pastas de trabalho mais seguras e estáveis, não pode ficar de fora disso. Mais comumente, os programadores profissionais usam o Git junto com uma plataforma na Web, como GitHub, GitLab, Bitbucket ou Azure DevOps. Essas plataformas permitem que você trabalhe com os chamados *pull requests* ou *merge requests*. Eles permitem que os desenvolvedores solicitem formalmente que suas alterações sejam mescladas na base de código principal. Um pull request oferece as seguintes informações:

- Quem é o autor das alterações.
- Quando foram feitas as alterações.

6 Disponível em https://www.xlwings.org/blog/unittests-for-microsoft-excel (conteúdo em inglês).