

Visual Studio | Documentação do Python

Tutoriais e artigos conceituais sobre o desenvolvimento de aplicativos em Python com o Visual Studio.

Introdução ao Python no Visual Studio

VISÃO GERAL

[Trabalhar com Python no Visual Studio no Windows](#)

COMEÇAR AGORA

[Instalar o Python no Visual Studio](#)

[Escrever e editar seu código](#)

[Depurar seu código](#)

[Escrever e executar testes](#)

[Usar REPL interativo](#)

[Gerenciar ambientes do Python](#)

[Escrever extensões C/C++ para o Python](#)

Desenvolvimento Web do Python

TUTORIAL

[Começar a usar o Python no Visual Studio](#)

[Criar aplicativos Web com Django](#)

[Criar aplicativos Web com Flask](#)

Supporte do Python no Visual Studio no Windows

Artigo • 18/04/2024

O Python é uma linguagem de programação popular que é confiável, flexível, fácil de aprender, de uso gratuito em todos os sistemas operacionais e respaldada por uma sólida comunidade de desenvolvedores e várias bibliotecas gratuitas. O Python oferece suporte a todos os tipos de desenvolvimento, incluindo aplicativos Web, serviços Web, aplicativos da área de trabalho, scripts e computação científica. Cientistas, desenvolvedores casuais, desenvolvedores profissionais e muitas universidades usam o Python para programação. Saiba mais sobre a linguagem em [python.org](#) e em [Python para iniciantes](#).

O Visual Studio é um IDE do Python poderoso no Windows. O Visual Studio é compatível com [software livre](#) para a linguagem Python por meio de cargas de trabalho de **Desenvolvimento em Python** e de **Ciência de Dados** (Visual Studio 2017 e posteriores) e a extensão gratuita das Ferramentas Python para Visual Studio (Visual Studio 2015 e anteriores). [Faça um tour pelo IDE do Visual Studio](#) e familiarize-se com os recursos do IDE para escrita e edição de código em Python.

O Visual Studio Code está disponível para Mac e Linux. Para saber mais, confira [Perguntas e respostas](#).

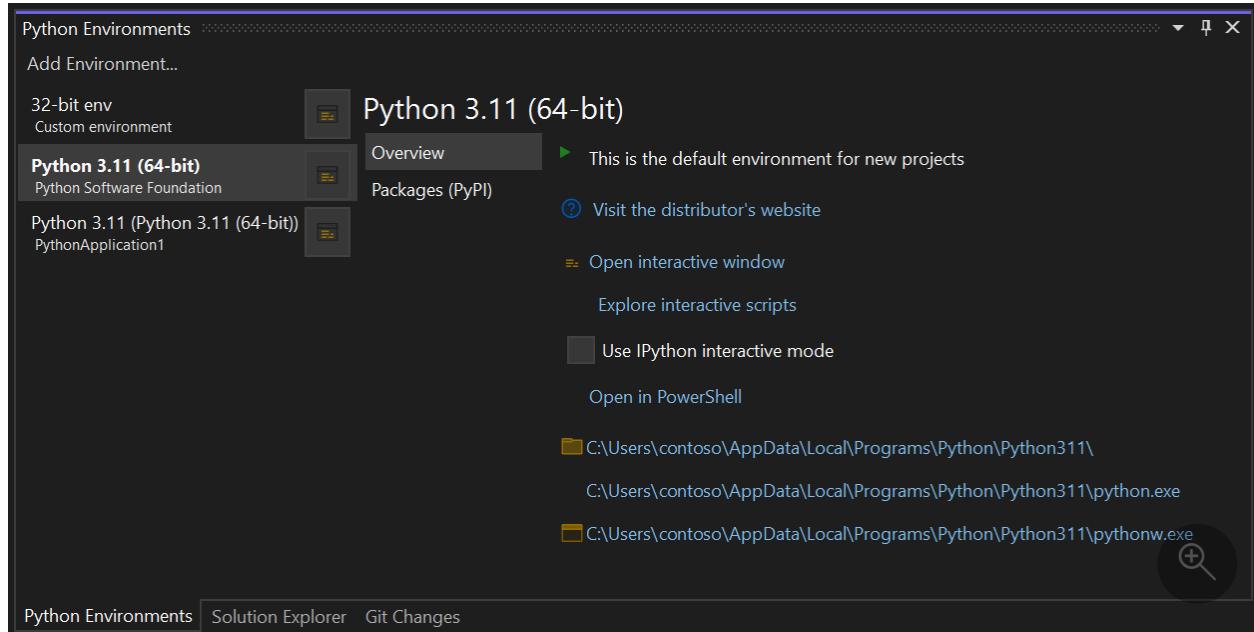
Introdução:

- Siga as [instruções de instalação](#) para configurar a carga de trabalho Python.
- Familiarize-se com os recursos do Python no Visual Studio examinando as seções neste artigo.
- Realize ou mais Guias de Início Rápido para criar um projeto. Se não tiver certeza, comece com [Início rápido: Abrir e executar código Python em uma pasta](#) ou [Criar um aplicativo Web com Flask](#).
- Siga o tutorial [Trabalhar com o Python no Visual Studio](#) para ter uma experiência completa de ponta a ponta.

Supporte para vários interpretadores

A janela **Ambientes do Python** do Visual Studio fornece um único local para gerenciar todos os ambientes do Python globais, os ambientes do Conda e os ambientes virtuais.

O Visual Studio detecta automaticamente as instalações do Python em locais padrão e permite que você configure instalações personalizadas. Em cada ambiente, é possível gerenciar pacotes, abrir uma janela interativa desse ambiente e acessar as pastas do ambiente facilmente.



Use o comando **Abrir janela interativa** para executar o Python de maneira interativa no contexto do Visual Studio. Use o comando **Abrir no PowerShell** para abrir uma janela Comando separada na pasta do ambiente selecionado. Nessa janela de comando, você pode executar qualquer script do Python.

Para mais informações:

- [Gerenciar ambientes do Python](#)
- [Referência aos ambientes do Python](#)

Edição avançada, IntelliSense e compreensão do código

O Visual Studio oferece um editor de Python de primeira classe, incluindo coloração de sintaxe, preenchimento automático em todo o código e em todas as bibliotecas, formatação de código, ajuda de assinatura, refatoração, dicas de tipo e linting. O Visual Studio também fornece recursos exclusivos como modo de exibição de classe, **Ir para Definição**, **Localizar Todas as Referências** e snippets de código. A integração direta com a **janela Interativa** ajuda você a desenvolver rapidamente um código em Python existente em um arquivo.

A screenshot of a Python code editor showing code completion. The code defines a function `main` that iterates over dates and creates a map of the solar system. The cursor is at `ss_map.a`, and a dropdown menu shows suggestions: `add_planet` (highlighted in blue), `data`, and `date`.

```
def main():
    for date in daterange(2015, 2016, step=timedelta(days=7)):
        # Create a map of the solar system on a particular day
        ss_map = SolarSystemMap(100, 27, date)
        ss_map.a|
```

Para mais informações:

- [Editar o código Python](#)
- [Código de formatação](#)
- [Refatorar o código](#)
- [Usar um linter](#)
- [Recursos do editor de código](#)

Janela Interativa

Para cada ambiente do Python conhecido para o Visual Studio, você pode abrir facilmente o mesmo ambiente interativo (REPL) de um interpretador de Python diretamente no Visual Studio, em vez de usar um prompt de comando separado. Também é possível mudar facilmente de ambiente. Para abrir um prompt de comando separado, selecione o ambiente desejado na janela **Ambientes do Python** e escolha o comando **Abrir no PowerShell**, conforme explicado anteriormente na seção [Suporte para vários interpretadores](#).

A screenshot of the Python Application Interactive window. It shows a command-line session where the user imports the `math` module and calculates sines for values -0.5, 0, 0.5, and 1.0. The output is a list: `[-1.0, 0.0, 1.0, 1.2246467991473532e-16]`. The window has tabs for `Error List` and `Output`.

```
PythonApplication1 Interactive 1
Environment: PythonApplication1 Module: __main__
>>> import math
>>> sines = []
>>> for i in [-0.5, 0, 0.5, 1.0]:
...     sines.append(math.sin(math.pi*i))
...
>>> i
1.0
>>> sines
[-1.0, 0.0, 1.0, 1.2246467991473532e-16]
>>>
```

O Visual Studio também fornece uma forte integração entre o editor de código Python e a janela **Interativa**. Para facilitar, o atalho de teclado **Ctrl+Enter** envia a linha de código (ou bloco de código) atual no editor para a janela **Interativa** e passa para a próxima linha (ou bloco). **Ctrl+Enter** permite percorrer o código facilmente sem precisar

executar o depurador. Também é possível enviar o código escolhido para a janela **Interativa** com o mesmo pressionamento de tecla e colar o código facilmente da janela **Interativa** para o editor. Juntos, esses recursos permitem que você elabore detalhes de um segmento de código na janela **Interativa** e salve os resultados facilmente em um arquivo no editor.

O Visual Studio também é compatível com IPython/Jupyter no REPL, incluindo gráficos embutidos, .NET e WPF (Windows Presentation Foundation).

Para mais informações:

- [Janela Interativa do Python](#)
- [IPython no Visual Studio](#)

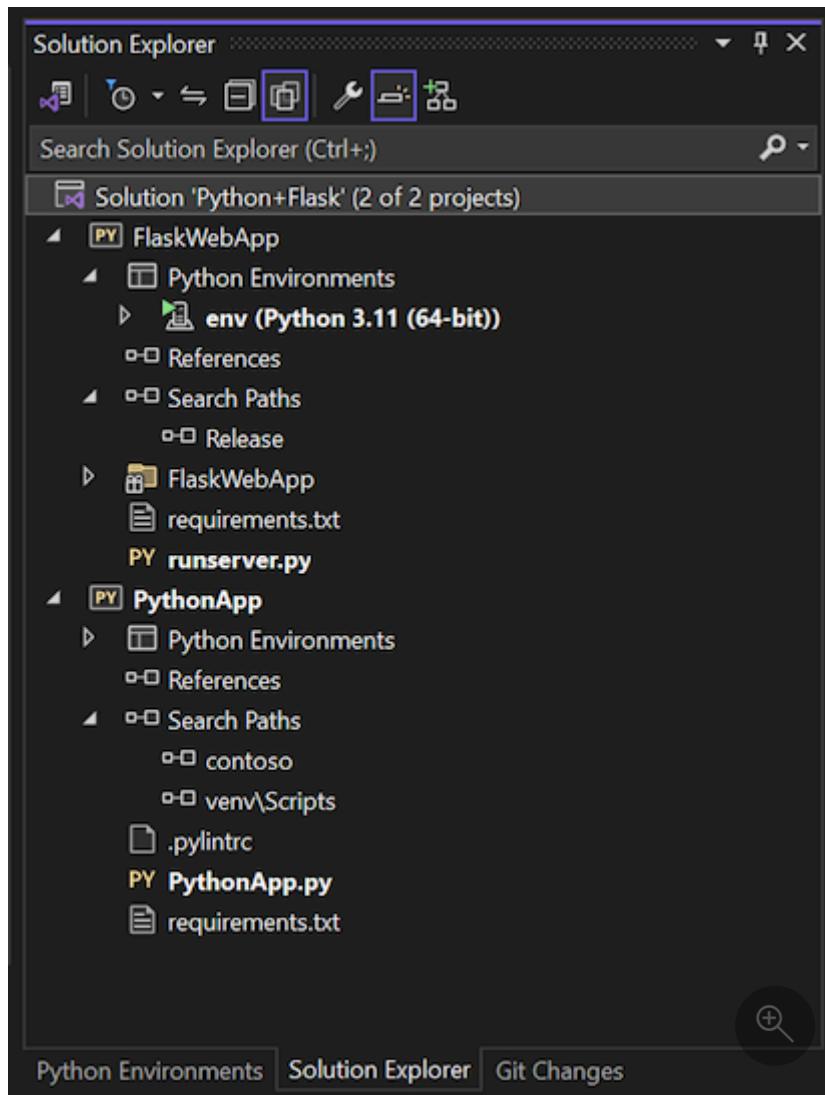
Sistema de projeto e modelos de projeto e de item

O Visual Studio ajuda você a gerenciar a complexidade de um projeto à medida que ele cresce ao longo do tempo. Um *projeto* do Visual Studio é muito mais do que uma simples estrutura de pastas. Um projeto auxilia no reconhecimento de como diferentes arquivos são utilizados e como eles se relacionam entre si. O Visual Studio ajuda a diferenciar código do aplicativo, código de teste, páginas da Web, JavaScript, scripts de build e assim por diante, o que permite usar os recursos apropriados para cada arquivo. Uma *solução* do Visual Studio ajuda você a gerenciar vários projetos relacionados, como um projeto do Python e um projeto de extensão em C++.

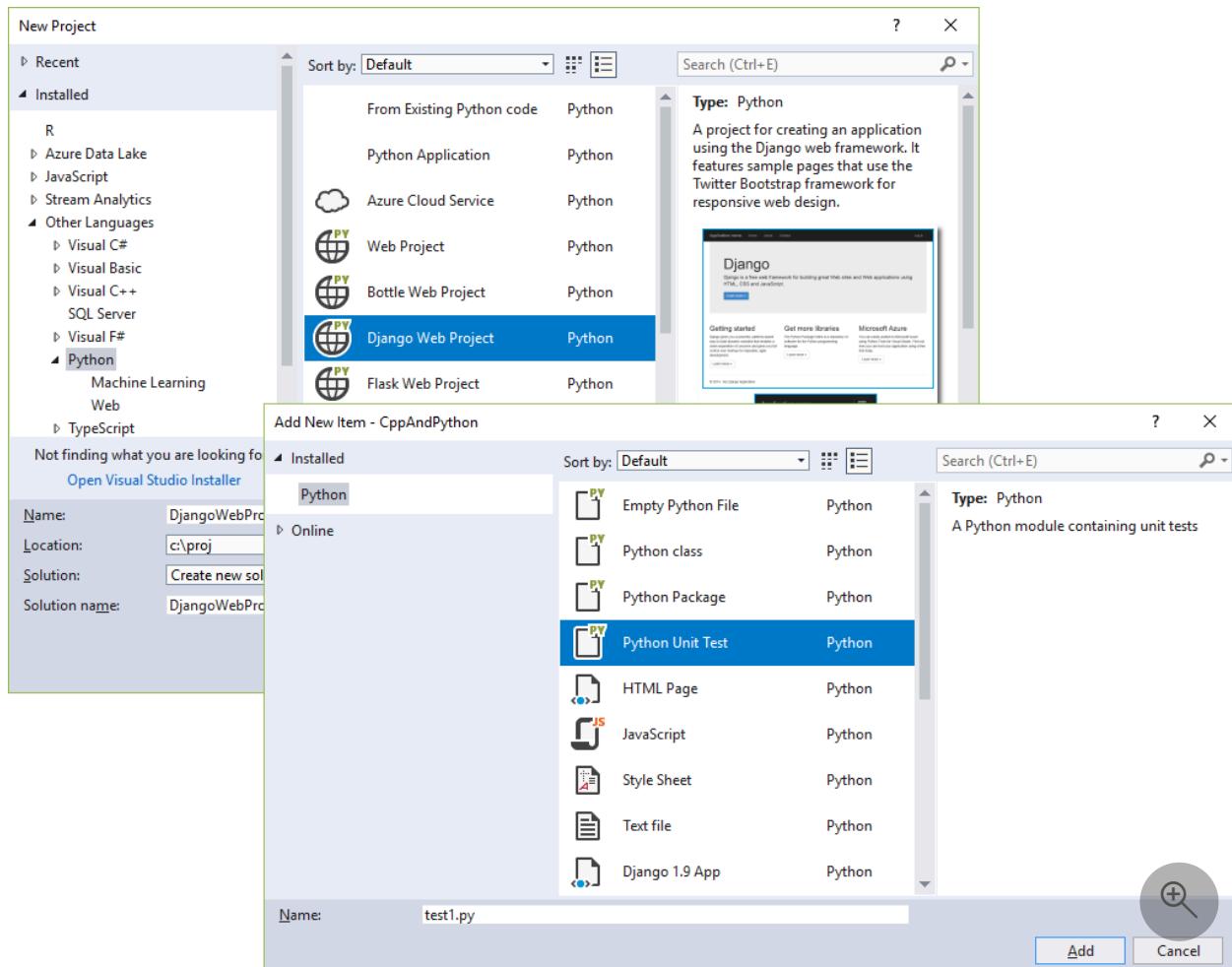
ⓘ Observação

No Visual Studio 2019 e em versões posteriores, você pode abrir uma pasta que contém código em Python e executar esse código sem criar um arquivo de projeto ou de solução do Visual Studio. Para obter mais informações, confira [Início Rápido: Abrir e executar código Python em uma pasta](#). Lembre-se de que há benefícios em usar um arquivo de projeto, conforme explicado nesta seção.

A imagem abaixo apresenta um exemplo de solução do Visual Studio com projetos do Python e do Flask no **Gerenciador de Soluções**.



Os modelos de projeto e de item automatizam o processo de configuração de diversos tipos de projeto e arquivo. Os modelos proporcionam uma economia de tempo valiosa e pouparam você do gerenciamento de detalhes complexos e propensos a erros. O Visual Studio oferece modelos para web, Azure, ciência de dados, console e outros tipos de projeto. Há modelos disponíveis para diversos arquivos, como classes do Python, testes de unidade, configuração do Azure para web, HTML e até mesmo aplicativos do Django.

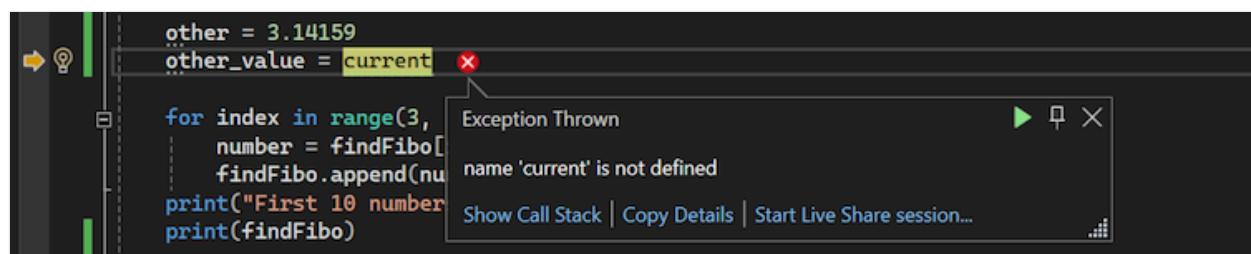


Para mais informações:

- [Gerenciar projetos do Python](#)
- [Referência de modelos de item](#)
- [Modelos de projeto do Python](#)
- [Trabalhar com o C++ e o Python](#)
- [Criar modelos de projeto e de item](#)
- [Soluções e projetos no Visual Studio](#)

Depuração completa

Um dos pontos fortes do Visual Studio é seu depurador avançado. Para Python especificamente, o Visual Studio inclui [depuração de modo misto](#) do Python/C++, depuração remota no Linux, depuração dentro da janela **Interativa** e depuração de testes de unidade do Python.



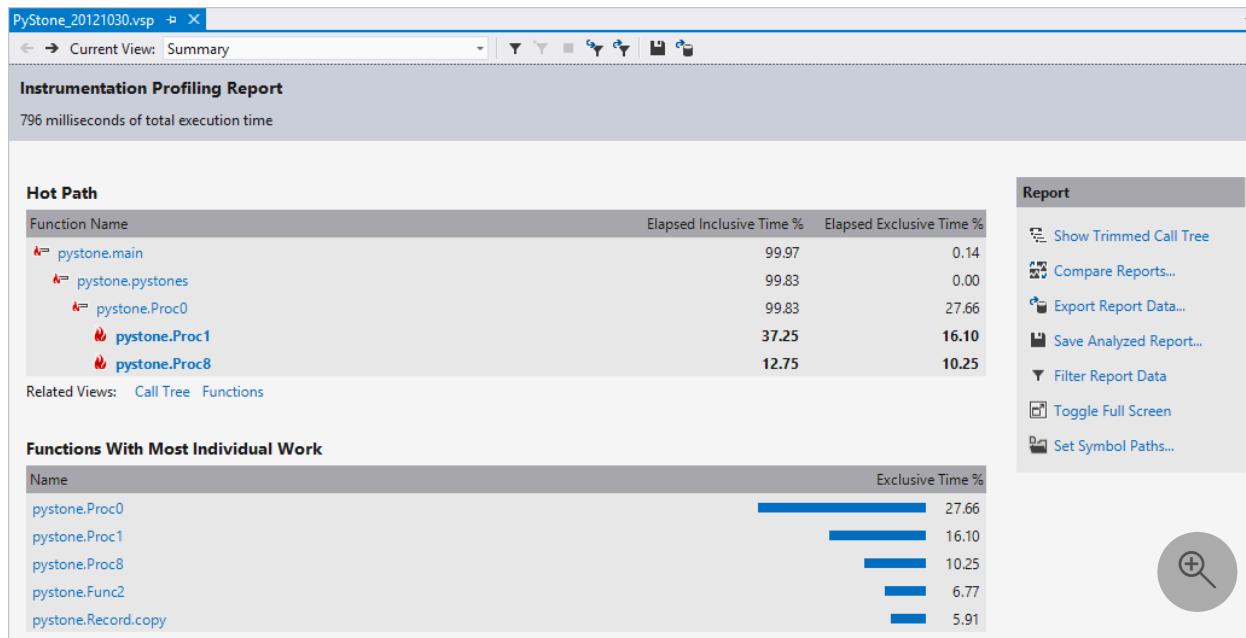
No Visual Studio 2019 e em versões posteriores, é possível executar e depurar código sem a necessidade de um arquivo de projeto do Visual Studio. Consulte [Início Rápido: abrir e executar o código do Python em uma pasta](#) para conferir um exemplo.

Para mais informações:

- [Depurar o Python](#)
- [Depuração de modo misto do Python/C++](#)
- [Depuração remota no Linux](#)
- [Tour de funcionalidades do depurador do Visual Studio](#)

Ferramentas de criação de perfil com relatórios abrangentes

A criação de perfil explora como o tempo está sendo gasto no aplicativo. O Visual Studio permite a criação de perfil com interpretadores baseados em CPython e inclui a capacidade de comparar o desempenho entre diferentes execuções de criação de perfil.



Para mais informações:

- [Ferramentas de criação de perfil do Python](#)
- [Tour pelos recursos de criação de perfil no Visual Studio](#)

Ferramentas de teste de unidade

Descubra, execute e gerencie testes no **Gerenciador de Testes** do Visual Studio e depure testes de unidade com facilidade.

The screenshot shows a Python test script in a code editor. A breakpoint is reached at the line `os.chdir("C:\\Windows")`. The code then continues to the assertion `self.assertNotEqual(cwd, new_cwd)`. Below the code is a 'Watch 1' window displaying two variables: `cwd` with value 'C:\\\\Users\\\\[REDACTED]' and `new_cwd` with value 'C:\\Windows'. The 'Type' column shows both are of type str.

```
class TestOS(unittest.TestCase):
    def test_os_chdir(self):
        cwd = os.getcwd()
        print("Current dir: {}".format(cwd))

        self.addCleanup(os.chdir, cwd)
        os.chdir("C:\\Windows")
        new_cwd = os.getcwd()

        self.assertNotEqual(cwd, new_cwd)
```

| Name | Value | Type |
|---------|-----------------------------|------|
| cwd | 'C:\\\\Users\\\\[REDACTED]' | str |
| new_cwd | 'C:\\Windows' | str |

Para mais informações:

- [Ferramentas de teste de unidade do Python](#)
- [Realizar teste de unidade do seu código](#)

SDK do Azure para Python

As bibliotecas do Azure para Python simplificam o consumo de serviços do Azure em aplicativos do Windows, do macOS X e do Linux. Você pode usá-las para criar e gerenciar recursos do Azure, e para se conectar aos serviços do Azure.

Para obter mais informações, confira [SDK do Azure para Python](#) e [Bibliotecas do Azure para Python](#).

Perguntas e respostas

Q. O suporte para Python está disponível com o Visual Studio para Mac?

R. Ainda não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

P. O que pode ser usado para criar a interface do usuário com o Python?

R. A oferta principal nessa área é o [Projeto Qt](#), com associações para Python conhecidas como [PySide \(a associação oficial\)](#) (consulte também [Downloads do PySide](#)) e [PyQt](#). O suporte do Python no Visual Studio não inclui quaisquer ferramentas específicas para desenvolvimento da interface do usuário.

P. Um projeto do Python pode produzir um executável autônomo?

R. Geralmente, o Python é uma linguagem interpretada, na qual o código é executado sob demanda em um ambiente compatível com o Python, como o Visual Studio e servidores Web. O Visual Studio ainda não fornece meios para criar um executável autônomo, o que, basicamente, é um programa com um interpretador de Python incorporado. No entanto, a comunidade do Python oferece maneiras diferentes de criar executáveis, conforme descrito em [StackOverflow](#). O CPython também dá suporte a ser inserido em um aplicativo nativo, conforme descrito na postagem do blog [Usar o arquivo .zip que permite inserção do CPython](#).

Conteúdo relacionado

- [Trabalhar com o Python no Visual Studio](#)
- [Início Rápido: abrir e executar o código do Python em uma pasta](#)
- [Janela Interativa do Python](#)

Comentários

Esta página foi útil?

 Yes

 No

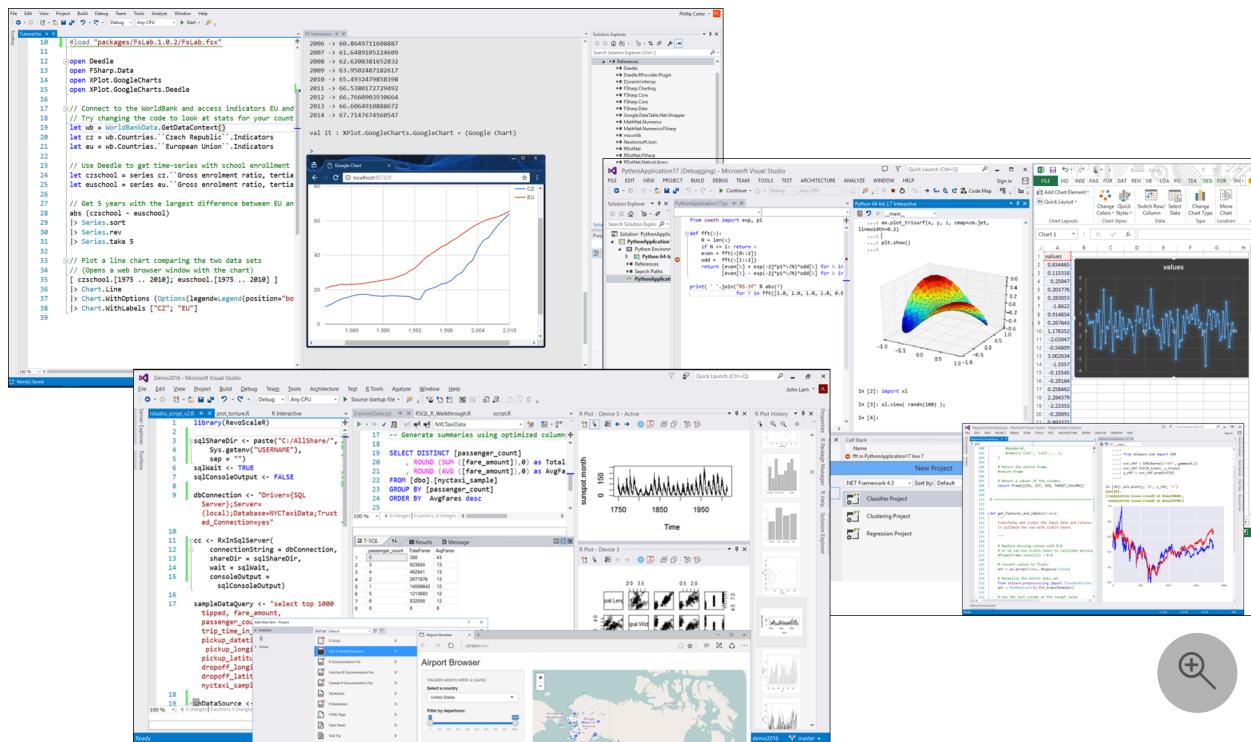
Carga de trabalho para aplicativos de ciência de dados e análise no Visual Studio

Artigo • 18/04/2024

A carga de trabalho para aplicativos de ciência de dados e de análise no Visual Studio reúne várias linguagens e suas respectivas distribuições de runtime:

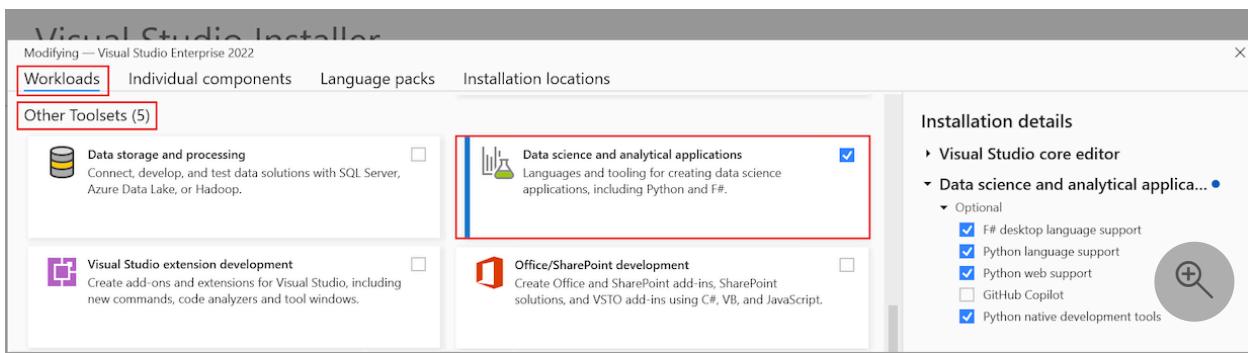
- Python
- F# com .NET Framework

O Python é uma das linguagens de script principais usadas para ciência de dados. O Python é fácil de aprender e tem o suporte de um rico ecossistema de pacotes. Esses pacotes abordam uma ampla variedade de cenários, como aquisição de dados, limpeza, treinamento de modelo, implantação e criação de gráficos. O F# também é uma linguagem do .NET robusta e funcional, adequada a diversas tarefas de processamento de dados.



Instalação da carga de trabalho

A carga de trabalho para aplicativos de ciência de dados e análise está disponível no instalador do Visual Studio em **Cargas de Trabalho>Outros conjuntos de ferramentas**:



Por padrão, a carga de trabalho instala as seguintes opções, que você pode modificar na seção de resumo da carga de trabalho no Instalador do Visual Studio:

- Suporte à linguagem F# da área de trabalho
- Python:
 - Suporte da linguagem Python
 - Suporte Web do Python
 - Ferramentas de desenvolvimento nativo do Python

Integração ao SQL Server

O SQL Server dá suporte ao uso do Python para análises avançadas diretamente no SQL Server. O suporte ao Python está disponível no SQL Server 2017 CTP 2.0 e posterior.

Aproveite as seguintes vantagens executando o código no local em que os dados já residem:

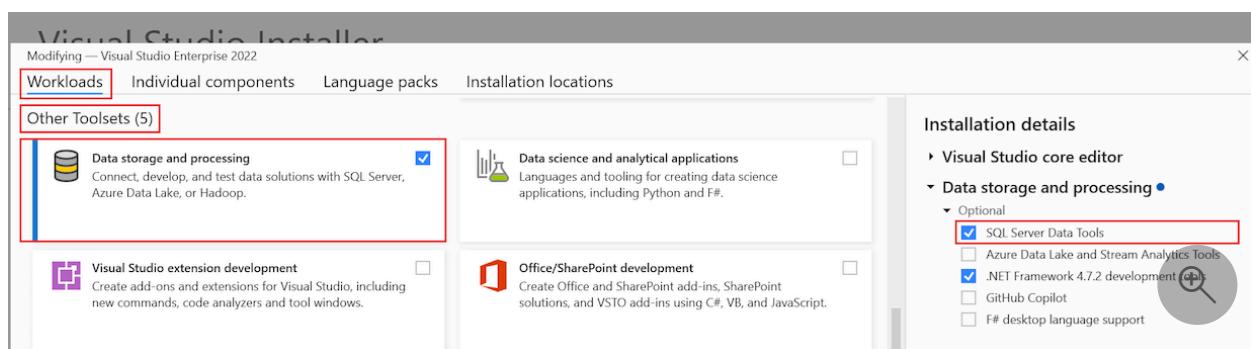
- **Eliminação da movimentação de dados:** em vez de mover dados do banco de dados para o aplicativo ou modelo, é possível compilar aplicativo ou modelo no banco de dados. Essa funcionalidade elimina os obstáculos de segurança, conformidade, governança e integridade, bem como uma série de problemas semelhantes relacionados à movimentação de grandes quantidades de dados. Também é possível consumir conjuntos de dados que não se ajustam à memória de uma máquina cliente.
- **Fácil implantação:** depois de ter um modelo pronto, implantá-lo na produção é uma simples questão de incorporar o modelo em um script de T-SQL. Os aplicativos cliente SQL codificados em qualquer linguagem poderão aproveitar os modelos e a inteligência por meio de uma chamada de procedimento armazenado. Nenhuma integração de linguagem específica é necessária.
- **Desempenho e escala de nível empresarial:** você pode usar as funcionalidades avançadas do SQL Server, como tabelas na memória e índices de armazenamento de colunas, com as APIs escalonáveis de alto desempenho nos pacotes RevoScale.

Eliminar a movimentação de dados também significa evitar restrições de memória do cliente à medida que os dados aumentam ou que você deseja aumentar o desempenho do aplicativo.

- **Extensibilidade avançada:** você pode instalar e executar quaisquer pacotes de software livre no SQL Server para compilar aplicativos de aprendizado profundo e de inteligência artificial em grandes quantidades de dados no SQL Server. A instalação de um pacote no SQL Server é tão simples quanto instalar um pacote no computador local.
- **Ampla disponibilidade sem custo adicional:** as integrações ade linguagem estão disponíveis em todas as edições do SQL Server 2017 e posterior, incluindo a edição Express.

Instalação de integração do SQL Server

Para aproveitar ao máximo a integração ao SQL Server, use o Instalador do Visual Studio para instalar a carga de trabalho em **Cargas de trabalho > Outros Conjuntos de Ferramentas > Armazenamento e processamento de dados**. Selecione a opção **SQL Server Data Tools** para habilitar o destaque de sintaxe e a implantação do SQL IntelliSense.



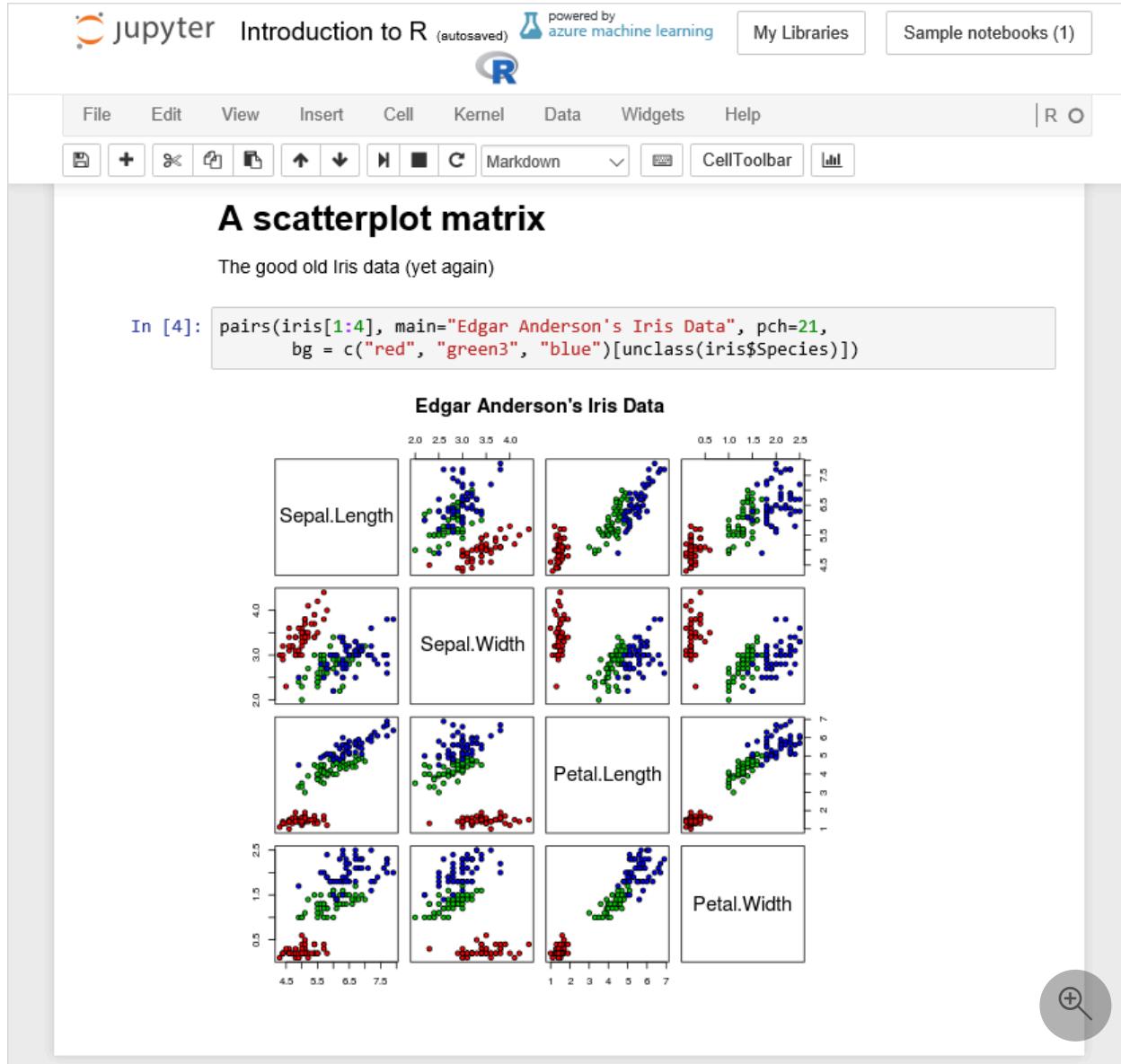
Para obter mais informações, confira [Python in SQL Server 2017: Enhanced in-database machine learning \(blog\)](#).

Outros serviços e SDKs

Além do que está diretamente na carga de trabalho para aplicativos de ciência de dados e de análise, o Notebooks no Visual Studio Code e o SDK do Azure para Python também são úteis para ciência de dados.

O SDK do Azure para Python facilita o consumo e gerenciamento de serviços do Microsoft Azure em aplicativos executados no Windows, Mac e Linux. Para obter mais informações, confira [Azure para desenvolvedores Python](#).

Você pode combinar a extensão do Jupyter com Notebooks no Visual Studio Code para dar suporte ao desenvolvimento do Jupyter e aprimorar seu projeto com extensões de linguagem extras. Como introdução, o serviço inclui blocos de anotações de exemplo em Python, em R e em F#. Para obter mais informações, confira [Experiências de notebooks da Microsoft e GitHub ↗](#).



Comentários

Esta página foi útil?

Yes

No

Instalar o suporte ao Python no Visual Studio

Artigo • 18/04/2024

Atualmente, o suporte ao Python está disponível apenas no Visual Studio para Windows. No Mac e no Linux, o suporte ao Python está disponível por meio do [Visual Studio Code](#).

Pré-requisitos

- Visual Studio no Windows. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).

Baixar e instalar a carga de trabalho do Python

Conclua as etapas a seguir para baixar e instalar a carga de trabalho do Python.

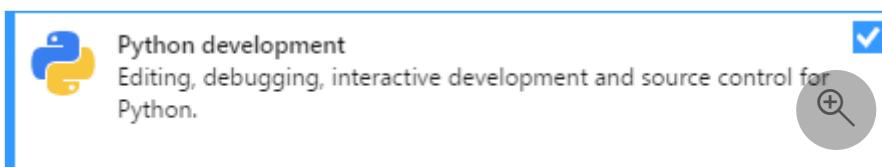
1. Baixe e execute o Instalador do Visual Studio mais recente para Windows. O suporte do Python está presente na versão 15.2 e posterior. Se você já tiver o Visual Studio instalado, abra-o e execute o instalador selecionando **Ferramentas>Obter Ferramentas e Recursos**.

[Instalar o Visual Studio Community](#)

Dica

A edição Community é para desenvolvedores individuais, aprendizado em sala de aula, pesquisa acadêmica e desenvolvimento de software livre. Para outros usos, instale o [Visual Studio Professional](#) ou o [Visual Studio Enterprise](#).

2. O instalador fornece uma lista de cargas de trabalho, que são grupos de opções relacionadas para áreas de desenvolvimento específicas. Para Python, selecione a carga de trabalho **Desenvolvimento do Python** e selecione **Instalar**:



| Opções de instalação do Python | Descrição |
|---|--|
| Distribuições do Python | Escolha qualquer combinação de distribuição do Python com a qual você planeja trabalhar. As opções comuns incluem variantes de 32 bits e 64 bits do Python 2, Python 3, Miniconda, Anaconda 2 e Anaconda 3. Cada opção inclui o interpretador, o runtime e as bibliotecas da distribuição. A Anaconda, especificamente, é uma plataforma de ciência de dados aberta que inclui uma ampla gama de pacotes pré-instalados. O Visual Studio automaticamente detecta as instalações existentes do Python. Para obter mais informações, confira Janela de ambientes do Python . Além disso, se uma versão mais recente do Python do que a versão mostrada no instalador estiver disponível, você poderá instalar a nova versão separadamente e o Visual Studio a detectará. |
| Suporte do modelo Cookiecutter | Instale a interface do usuário gráfica do Cookiecutter para descobrir modelos, inserir opções de modelo e criar projetos e arquivos. Para obter mais informações, confira Usar a extensão Cookiecutter . |
| Suporte Web do Python | Instale ferramentas para desenvolvimento na Web, incluindo suporte à edição HTML, CSS e JavaScript, juntamente com modelos para projetos que usam as estruturas Bottle, Flask e Django. Para obter mais informações, confira Modelos de projeto Web do Python . |
| Ferramentas de desenvolvimento nativo do Python | Instale o compilador do C++ e outros componentes necessários para desenvolver extensões nativas para Python. Para obter mais informações, confira Criar uma extensão do C++ para Python . Além disso, instale a carga de trabalho Desenvolvimento de área de trabalho C++ para obter suporte total a C++. |

Por padrão, a carga de trabalho do Python é instalada para todos os usuários em um computador em:

```
%ProgramFiles%\Microsoft Visual Studio\<VS_version>\<VS_edition>Common7\IDE\Extensions\Microsoft\Python
```

em que `<VS_version>` é 2022 e `<VS_edition>` é Community, Professional ou Enterprise.

Testar sua instalação

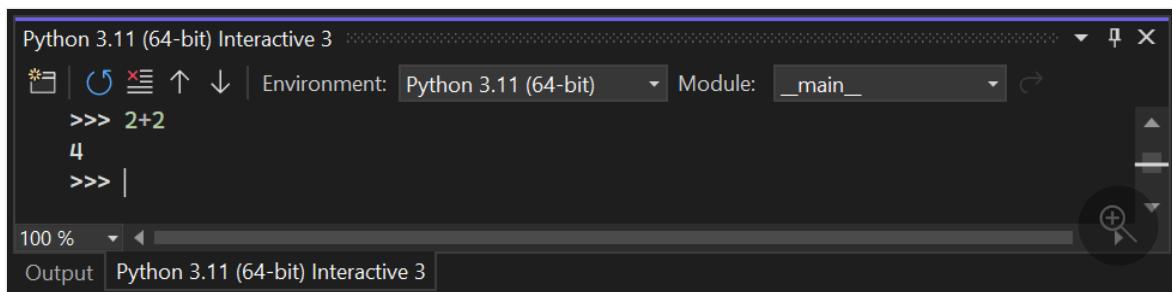
Verifique rapidamente a instalação do suporte do Python:

1. Inicie o Visual Studio.

2. Selecione Alt + I para abrir a janela **Python Interactive**.

3. Na janela, insira a instrução `2+2`.

A saída da instrução `4` é exibida na janela. Se você não vir a saída correta, verifique novamente as etapas.



The screenshot shows a Python interactive window titled "Python 3.11 (64-bit) Interactive 3". The window has a toolbar with icons for file operations, environment selection (set to "Python 3.11 (64-bit)"), and module selection (set to "_main_"). The main area displays the Python prompt ">>> 2+2" followed by the result "4". Below the main area is a status bar showing "100 %". At the bottom, there is a tab labeled "Output" and the title "Python 3.11 (64-bit) Interactive 3".

Conteúdo relacionado

- [Etapa 1 do tutorial: criar um projeto do Python](#)
- [Identificar manualmente um interpretador Python existente](#)

Comentários

Esta página foi útil?

 Yes

 No

Instalar interpretadores do Python

Artigo • 18/04/2024

Existem diversas opções para instalar interpretadores do Python e trabalhar com o Visual Studio. É possível instalar um interpretador no momento da instalação da carga de trabalho do Python ou depois que uma carga de trabalho estiver presente. Também é possível instalar interpretadores manualmente fora do Instalador do Visual Studio.

Quando você instala a carga de trabalho de desenvolvimento do Python no Visual Studio 2017 e em versões posteriores, o Python 3 (64 bits) também é instalado por padrão. Como opção, você pode optar por instalar a versão de 32 bits ou a versão de 64 bits do Python 2 ou do Python 3 com o Miniconda (Visual Studio 2019) ou o Anaconda 2/Apache 3 (Visual Studio 2017). As etapas para essa modalidade de instalação estão em [Instalar o suporte ao Python no Visual Studio](#).

Como alternativa, você pode instalar os interpretadores padrão do Python utilizando o recurso **Adicionar Ambiente** no Visual Studio. Essa opção está disponível na janela **Ambientes do Python** e na barra de ferramentas do Python.

Também é possível instalar os interpretadores do Python manualmente fora do Instalador do Visual Studio. Suponha que você instale o Anaconda 3 antes de instalar o Visual Studio. Não é necessário instalar o Anaconda novamente por meio do Instalador do Visual Studio. Também é possível instalar uma versão mais recente de um interpretador que ainda não esteja listada no Instalador do Visual Studio.

Pré-requisitos

- O Visual Studio dá suporte ao Python versão 3.7. Embora seja possível utilizar uma versão anterior do Visual Studio para editar código escrito em versões anteriores do Python, essas versões do Python não recebem suporte oficial. Alguns recursos do Visual Studio, como o IntelliSense e a depuração, podem não funcionar com versões anteriores do Python.
- Para o Visual Studio 2015 e versões anteriores, utilize o Python 3.5 ou versões anteriores. É necessário instalar manualmente um dos interpretadores do Python.

Distribuições do Anaconda

Embora o Visual Studio ofereça a instalação da distribuição do Anaconda, seu uso da distribuição e de outros pacotes do Repositório do Anaconda são associados pelos [Termos de Serviço do Anaconda](#). Esses termos podem exigir que algumas

organizações paguem ao Anaconda por uma licença comercial ou configurem as ferramentas para acessar um repositório alternativo. Para obter mais informações, consulte a [documentação de canais do Conda](#).

Listas de interpretadores do Python

A tabela a seguir lista os interpretadores do Python que podem ser utilizados com o Visual Studio.

 Expandir a tabela

| Interpretador | Descrição | Observações |
|----------------------------|--|--|
| CPython | O interpretador “nativo” e mais usado, disponível em versões de 32 e 64 bits (o recomendado é 32 bits). Inclui os últimos recursos de linguagem, a compatibilidade máxima com pacotes do Python, suporte de depuração completo e interoperabilidade com o IPython . Revise as considerações em Devo usar o Python 2 ou 3? para determinar qual versão do Python deve instalar. | O Visual Studio 2015 e versões anteriores não oferecem suporte ao Python 3.6 ou versões posteriores e podem gerar erros, como Python versão 3.6 sem suporte . Para o Visual Studio 2015 e versões anteriores, utilize o Python 3.5 ou versões anteriores. |
| IronPython | Uma implementação .NET do Python, disponível nas versões de 32 e 64 bits. Fornece interoperabilidade entre C#/F#/Visual Basic, acesso às APIs .NET, depuração padrão do Python (mas não depuração de modo misto em C++) e depuração mista em IronPython/C#. | O IronPython não oferece suporte a ambientes virtuais. |
| Anaconda | Uma plataforma aberta de ciência de dados baseada em Python. Inclui a versão mais recente do CPython e a maioria dos pacotes difíceis de instalar. | Se você não conseguir decidir qual interpretador usar, recomendamos o uso do Anaconda. |
| PyPy | Uma implementação JIT de rastreamento de alto desempenho do Python. Boa para programas de execução prolongada e situações em que são identificados problemas de desempenho, mas não são encontradas outras resoluções. | Funciona com o Visual Studio, mas com suporte limitado para recursos de depuração avançados. |

| Interpretador | Descrição | Observações |
|--------------------------|---|--|
| Jython 🔗 | Uma implementação do Python na JVM (Máquina Virtual Java). Semelhante ao IronPython, o código em execução no Jython pode interagir com classes e bibliotecas Java. No entanto, muitas das bibliotecas destinadas ao CPython podem não estar acessíveis. | Funciona com o Visual Studio, mas com suporte limitado para recursos de depuração avançados. |

Detectar o ambiente

O Visual Studio exibe todos os ambientes conhecidos na janela [Ambientes do Python](#). Ele detecta automaticamente as atualizações em interpretadores existentes.

Se o Visual Studio não detectar um ambiente instalado, consulte [Identificar manualmente um ambiente existente](#).

Se você quiser fornecer novas formas de detecção para ambientes do Python, consulte [Detecção para ambiente da PTVS](#) [🔗](#) (github.com).

Entradas do Registro

O Visual Studio (todas as versões) detecta automaticamente cada interpretador Python instalado e seu ambiente verificando o Registro, de acordo com o [PEP 514 – registro do Python no Registro do Windows](#) [🔗](#). Geralmente, as instalações do Python se encontram nas chaves **HKEY_LOCAL_MACHINE\SOFTWARE\Python** (32 bits) e **HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Python** (64 bits) nos nós da distribuição, como **PythonCore** (CPython) e **ContinuumAnalytics** (Anaconda).

Mover um interpretador

Se você mover um interpretador existente para um novo local usando o sistema de arquivos, o Visual Studio não detectará automaticamente a alteração.

- Se você especificou originalmente o local do interpretador por meio da janela **Ambientes do Python**, poderá editar o ambiente usando a guia **Configurar** nessa janela para identificar o novo local. Para obter mais informações, consulte [Identificar manualmente um ambiente existente](#).
- Se você instalou o interpretador usando um programa de instalação, use as etapas a seguir para reinstalar o interpretador no novo local:

1. Restaure o interpretador do Python para o local original.
2. Desinstale-o usando o instalador, que limpa as entradas do Registro.
3. Reinstale o interpretador no novo local.
4. Reinicie o Visual Studio, que deve detectar automaticamente o novo local em vez do local antigo.

Esse processo garante que as entradas do Registro que identificam o local do interpretador, usado pelo Visual Studio, sejam atualizadas corretamente. O uso de um instalador também lida com outros efeitos colaterais que possam existir.

Conteúdo relacionado

- [Gerenciar ambientes do Python](#)
- [Selecionar um interpretador para um projeto](#)
- [Usar requirements.txt para dependências](#)
- [Caminhos de pesquisa](#)
- [Referência da janela de ambientes do Python](#)

Comentários

Esta página foi útil?

 Yes

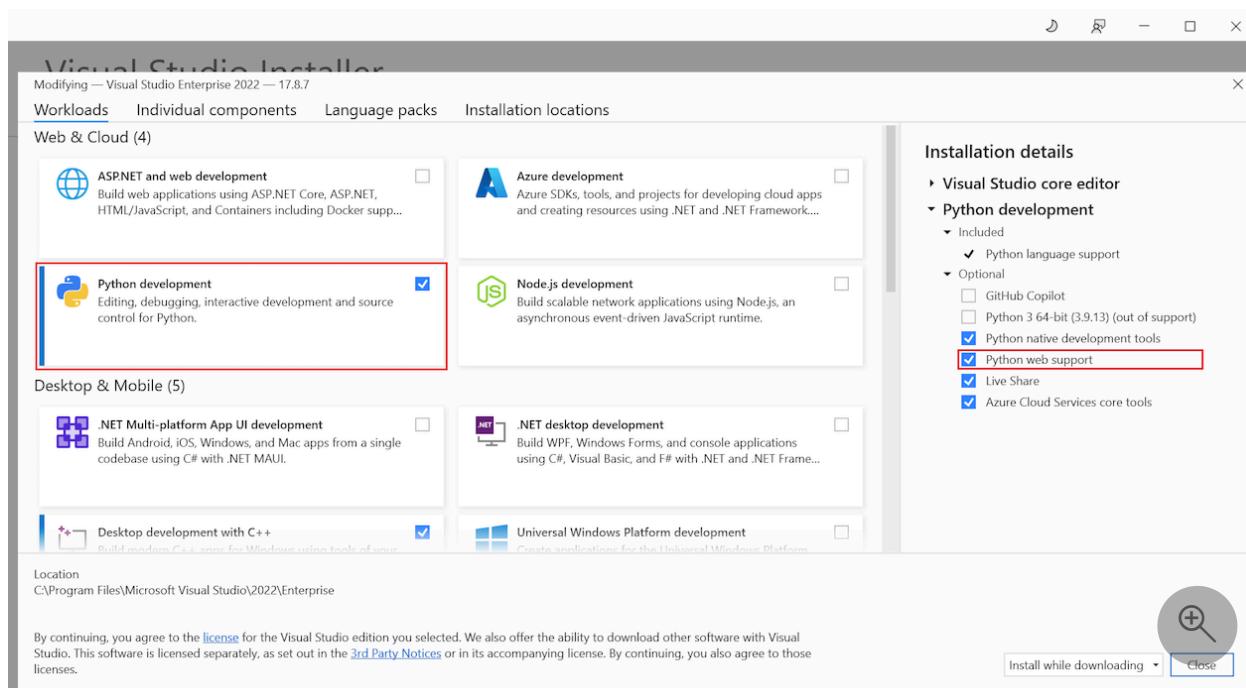
 No

Início rápido: criar o aplicativo Web com Visual Studio

Artigo • 18/04/2024

Neste guia de início rápido, você cria um aplicativo Web em Python com base na estrutura Flask no Visual Studio. Crie o projeto por meio de etapas simples para ajudarão a saber mais sobre os recursos básicos do Visual Studio. Explore como criar itens de projeto, adicionar código e executar aplicativos.

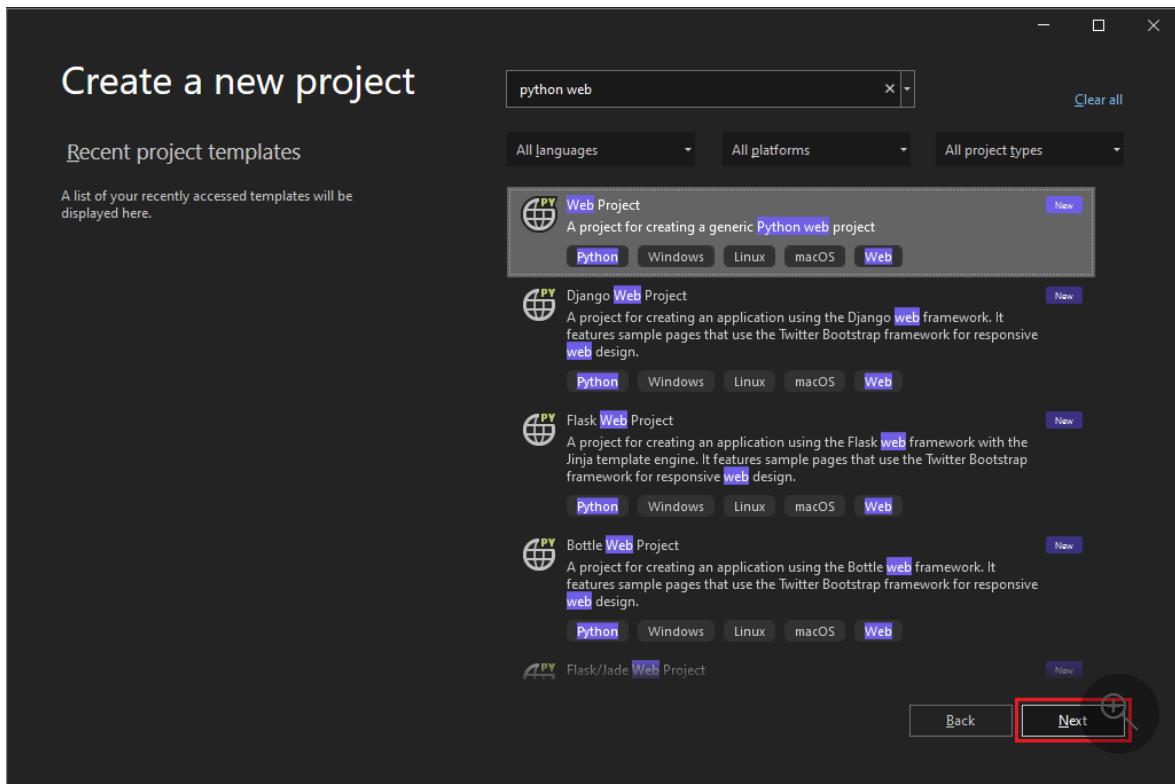
Se você precisar instalar o Visual Studio, acesse a página de [downloads do Visual Studio](#) para instalá-lo gratuitamente. No Instalador do Visual Studio, selecione a carga de trabalho de **desenvolvimento do Python** e, em detalhes da instalação, selecione **Supporte web do Python**.



Criar o projeto

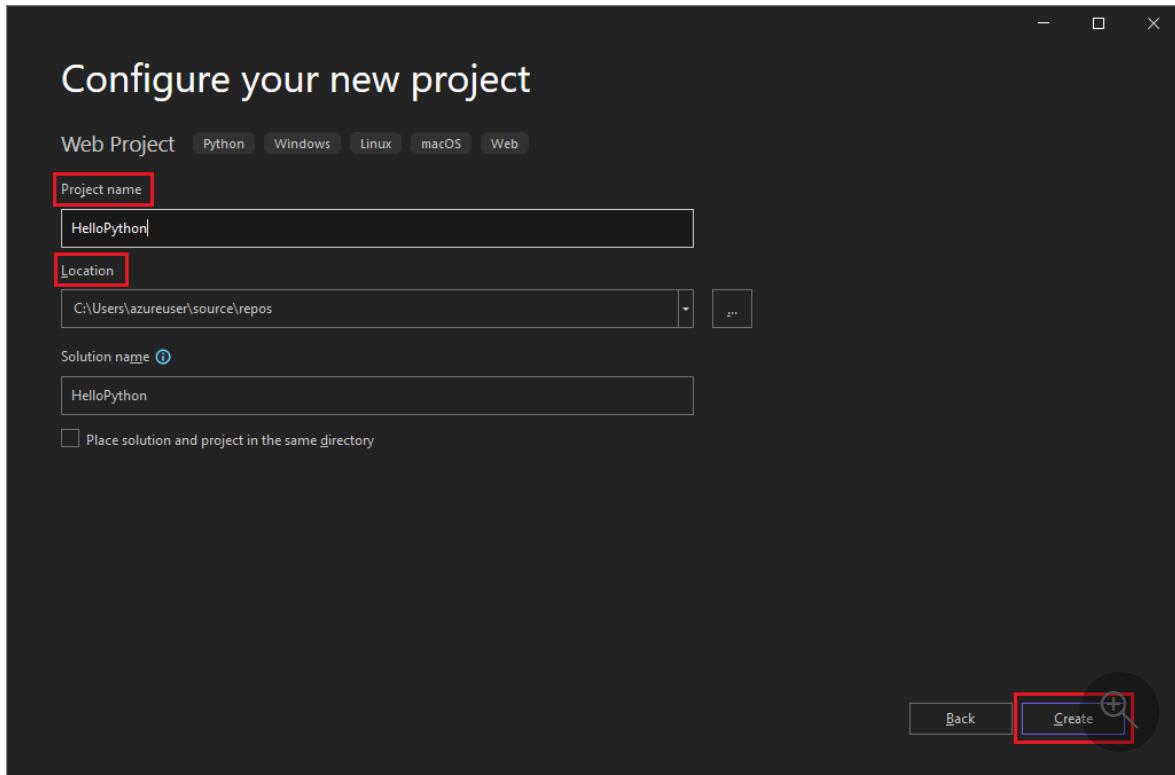
As seguintes etapas criam um projeto vazio que serve como um contêiner para o aplicativo.

1. Abra o Visual Studio. Na tela inicial, selecione **Criar um novo projeto**.
2. Na caixa de diálogo **Criar um novo projeto**, insira *Web Python* na caixa de pesquisa. Na lista de resultados, selecione **Projeto Web** e selecione **Avançar**.

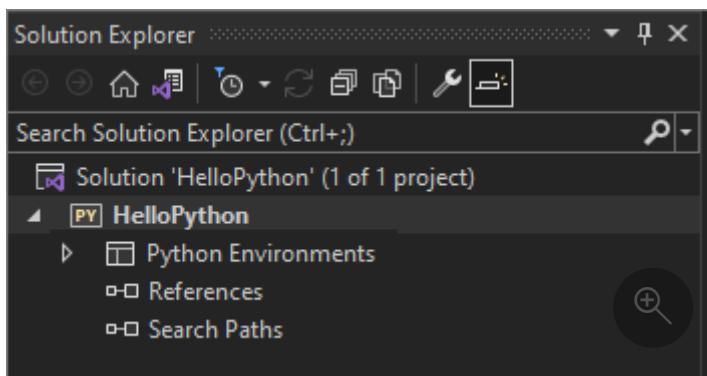


Se você não visualizar os modelos de projeto Web do Python, selecione **Ferramentas**>**Obter Ferramentas e Recursos** para executar o Instalador do Visual Studio. No instalador, selecione a carga de trabalho de **Desenvolvimento de Python**. Em **Detalhes da instalação**, selecione **Supporte à Web Python** e, em seguida, **Modificar**.

3. Na caixa de diálogo **Configurar seu novo projeto**, insira *HelloPython* para **Nome do projeto**, especifique um **Local** do projeto e selecione **Criar**.



O novo projeto é aberto no **Gerenciador de Soluções**. O **Nome da solução** é definido automaticamente para corresponder ao **Nome do projeto**. O novo projeto está vazio porque não contém nenhum arquivo.



Projetos e soluções no Visual Studio

Existem vantagens de criar um projeto no Visual Studio para um aplicativo do Python. Os aplicativos Python normalmente são definidos usando apenas pastas e arquivos, mas essa estrutura simples pode se tornar cara conforme o aumento dos aplicativos. Os aplicativos podem envolver arquivos gerados automaticamente, JavaScript para aplicativos Web e outros componentes. Um projeto do Visual Studio ajuda a gerenciar essa complexidade.

O projeto é identificado com um arquivo `.pyproj`, que identifica todos os arquivos de origem e de conteúdo associados ao seu projeto. O arquivo `.pyproj` contém informações de build de cada arquivo, mantém as informações para integração com sistemas de controle do código-fonte e ajuda a organizar a aplicação em componentes lógicos.

Uma solução do Visual Studio é um contêiner que ajuda você a gerenciar um ou mais projetos relacionados como um grupo. O Visual Studio mostra suas soluções no **Gerenciador de Soluções**. A solução armazena as configurações que não são específicas de um projeto. Os projetos em uma solução também podem fazer referência uns aos outros. Por exemplo, a execução de um projeto de aplicativo Python pode criar automaticamente um segundo projeto, como uma extensão C++ usada no aplicativo Python.

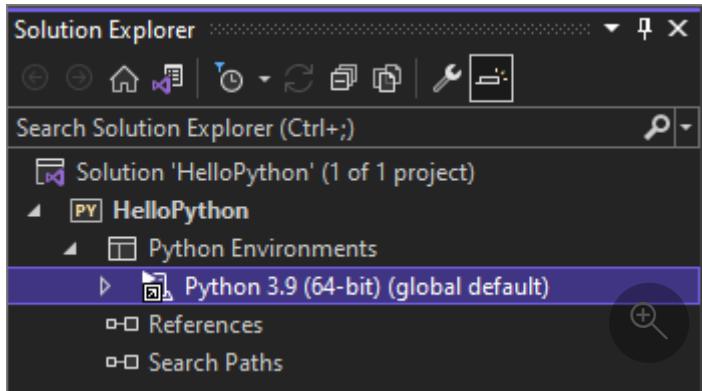
Instalar a biblioteca Flask

Aplicativos Web em Python quase sempre usam uma das muitas bibliotecas Python disponíveis para lidar com detalhes de baixo nível como o roteamento de solicitações da Web e formatação de respostas. O Visual Studio oferece muitos modelos para

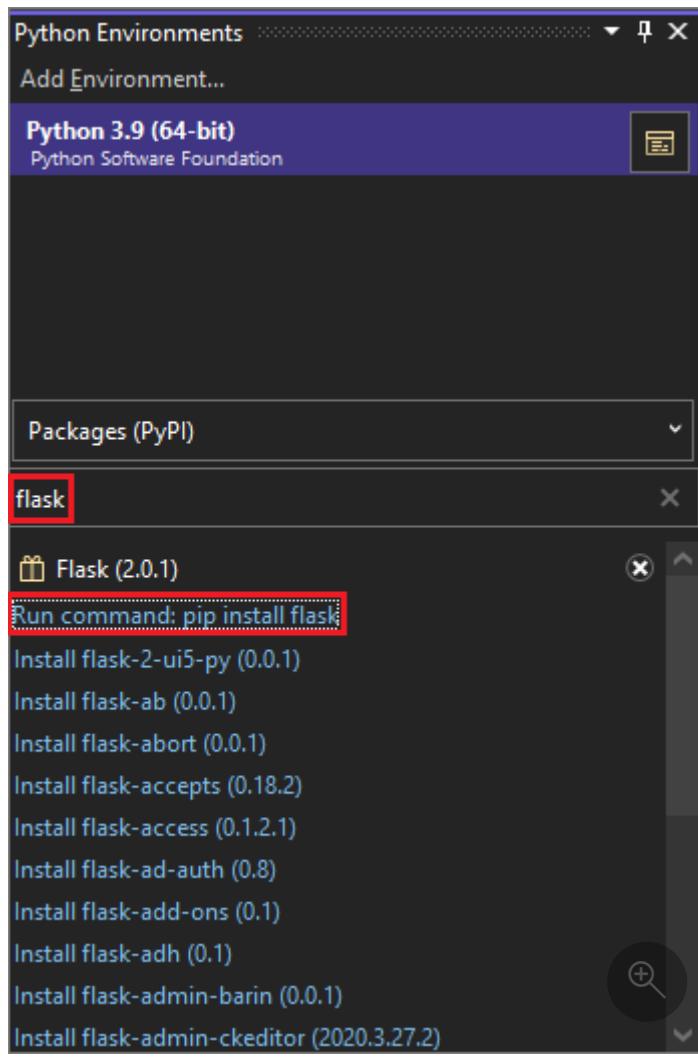
aplicativos Web. Você cria um projeto a partir de um desses modelos posteriormente neste Guia de início rápido.

Use as etapas a seguir para instalar a biblioteca Flask no *ambiente global* padrão que o Visual Studio usa para este projeto.

1. Expanda o nó **Ambiente do Python** no projeto para ver o ambiente padrão para o projeto.

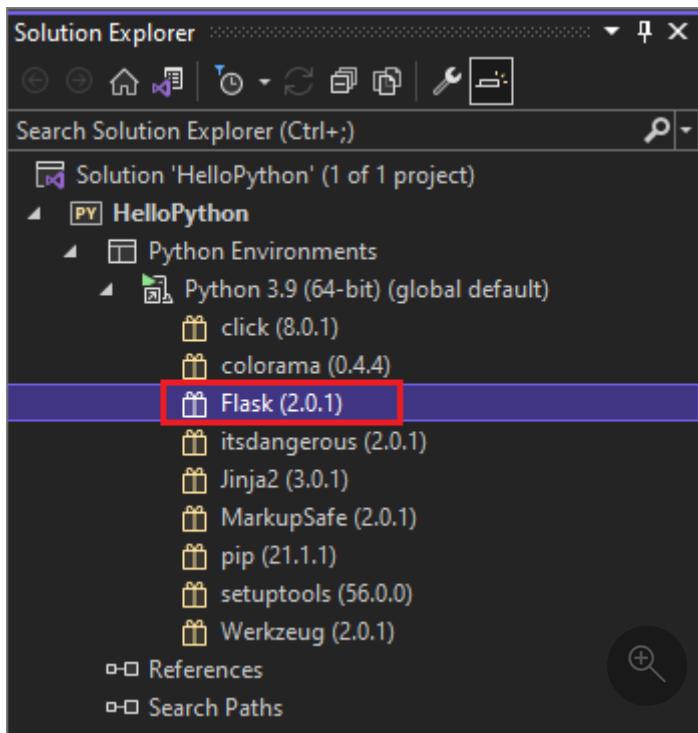


2. Clique com o botão direito do mouse no ambiente e selecione **Gerenciar Pacotes do Python**. Esse comando abre a janela **Ambientes de Python** na guia **Pacotes (PyPI)**.
3. Insira *flask* no campo de pesquisa.
 - Se o comando **Flask** aparecer abaixo da caixa de pesquisa, o Flask já está presente no sistema. Siga para a próxima etapa.
 - Se o comando **Flask** não aparecer abaixo da caixa de pesquisa, selecione **Executar comando: pip install flask**.



Um prompt de elevação aparece se a pasta de pacotes de ambiente global estiver em uma área protegida como C:\Arquivos de Programas. Aceite as solicitações de privilégios de administrador. Você pode observar a janela Saída do Visual Studio para progresso.

4. Depois de instalar o Flask, a biblioteca aparece no ambiente no **Gerenciador de Soluções**. Agora você pode usar comandos Flask em seu código Python.



⚠ Observação

Em vez de instalar as bibliotecas no ambiente global, os desenvolvedores geralmente criam um *ambiente virtual* no qual instalar bibliotecas para um projeto específico. Modelos do Visual Studio geralmente oferecem essa opção, conforme descrito em [Início Rápido: criar um projeto de Python usando um modelo](#).

Para obter mais informações sobre outros pacotes Python disponíveis, consulte o [Índice de pacotes Python](#).

Adicionar um arquivo de código

Agora você está pronto para adicionar algum código Python para implementar um aplicativo Web mínimo.

1. Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Adicionar>Novo Item**.
2. Na caixa de diálogo **Adicionar Novo Item**, selecione a opção de arquivo Python **Vazio**.
3. Insira o nome de arquivo *app.py* e selecione **Adicionar**. O Visual Studio abre o arquivo automaticamente em uma janela do editor.
4. Copie o código a seguir e cole no arquivo `app.py`:

Python

```
from flask import Flask

# Create an instance of the Flask class that is the WSGI application.
# The first argument is the name of the application module or package,
# typically __name__ when using a single module.
app = Flask(__name__)

# Flask route decorators map / and /hello to the hello function.
# To add other resources, create functions that generate the page
contents
# and add decorators to define the appropriate resource locators for
them.

@app.route('/')
@app.route('/hello')
def hello():
    # Render the page
    return "Hello Python!"

if __name__ == '__main__':
    # Run the app server on localhost:4449
    app.run('localhost', 4449)
```

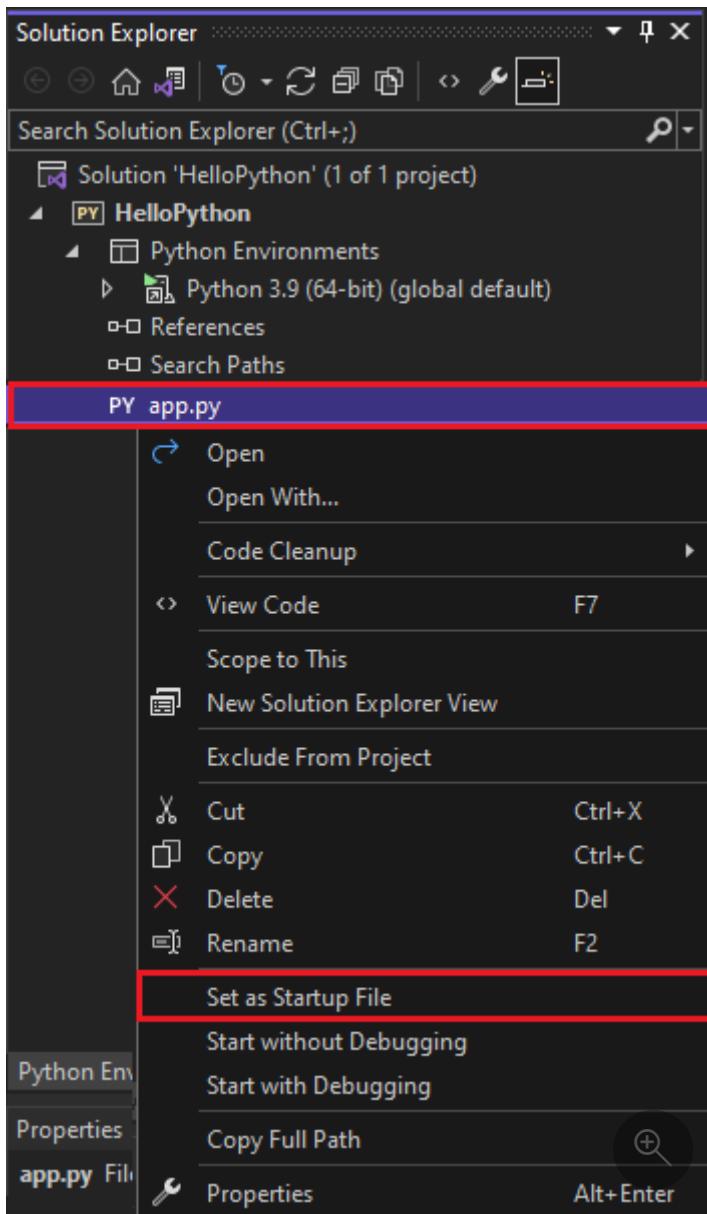
A caixa de diálogo **Adicionar Novo item** exibe muitos outros tipos de arquivos que você pode adicionar ao projeto do Python, como uma classe do Python, um pacote do Python, um teste de unidade do Python ou arquivos `web.config`. Em geral, esses *modelos de item* são uma ótima forma de criar rapidamente arquivos com código clichê útil.

Para obter mais informações sobre o Flask, consulte o [Guia de início rápido do Flask](#).

Executar o aplicativo

Para executar o aplicativo Web, siga estas etapas:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no arquivo `app.py` e selecione **Definir como arquivo de inicialização**. Esse comando identifica o arquivo de código para iniciar em Python ao executar o aplicativo.



2. Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Propriedades**.
3. Na caixa de diálogo **Propriedades**, na guia **Depurar**, defina a propriedade **Número da Porta** como `4449`. Essa definição garante que o Visual Studio inicie um navegador com `localhost:4449` para corresponder aos argumentos `app.run` no código.
4. No Visual Studio, selecione **Depurar>Iniciar sem depuração** ou **Ctrl+F5** para salvar as alterações nos arquivos e executar o aplicativo.

Uma janela de comando é aberta e exibe a mensagem `**Executando no https://localhost:4449**`. Uma janela do navegador é aberta `localhost:4449` e exibe a mensagem `Olá, Python!`. A solicitação `GET` também aparece na janela de comando com um status `200`.

Se um navegador não for aberto automaticamente, abra o navegador de sua preferência e vá até `localhost:4449`.

Caso apareça somente o shell interativo do Python na janela de comando ou se essa janela piscar brevemente na tela, verifique se o arquivo `app.py` está definido como arquivo de inicialização.

5. Na janela do navegador, vá até `localhost:4449/hello` para testar se o decorador do recurso `/hello` também funciona.

Mais uma vez, a solicitação `GET` aparece na janela de comando com um status de `200`.

Experimente algumas outras URLs para verificar se mostram os códigos de status `404` na janela de comando.

6. Feche a janela de comando para interromper o aplicativo e, em seguida, a janela do navegador.

Iniciar com ou sem depurar

Você pode executar seu aplicativo com ou sem depuração habilitada. Estas são as diferenças entre estas opções:

- O comando **Iniciar depuração** executa o aplicativo no contexto do [depurador do Visual Studio](#). Com o depurador, é possível definir pontos de interrupção, examinar variáveis e executar seu código linha por linha. Os aplicativos podem ser executados mais lentamente no depurador devido aos vários ganchos permitidos pela depuração.
- Use o comando **Iniciar sem depuração** para executar o aplicativo diretamente sem contexto de depuração, semelhante à execução do aplicativo a partir da linha de comando. Esse comando também inicia automaticamente um navegador e abre o URL especificado na guia **Propriedades>Depuração** do projeto.

Conteúdo relacionado

- [Início Rápido: Criar um projeto do Python com em modelo](#)
- [Tutorial: Introdução ao Python no Visual Studio](#)
- [Modelos de aplicativo Web Python no Visual Studio](#)

Comentários

Esta página foi útil?

 Yes

 No

Início Rápido: abrir e executar o código do Python em uma pasta no Visual Studio

Artigo • 18/04/2024

Neste guia de início rápido, você segue as etapas guiadas para executar o código do Python no Visual Studio 2019 e posterior sem precisar criar um projeto do Visual Studio. O Visual Studio facilita a abertura e a execução do código do Python existente a partir de uma pasta. Os mesmos recursos e comandos estão disponíveis para o desenvolvimento do código do Python, como quando você escolhe trabalhar com um projeto.

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- As instruções neste guia de início rápido se aplicam a qualquer pasta com código Python. Para seguir o exemplo descrito neste artigo, clone o repositório do GitHub `gregmalcolm/python_koans` para o computador, usando o seguinte comando:

Console

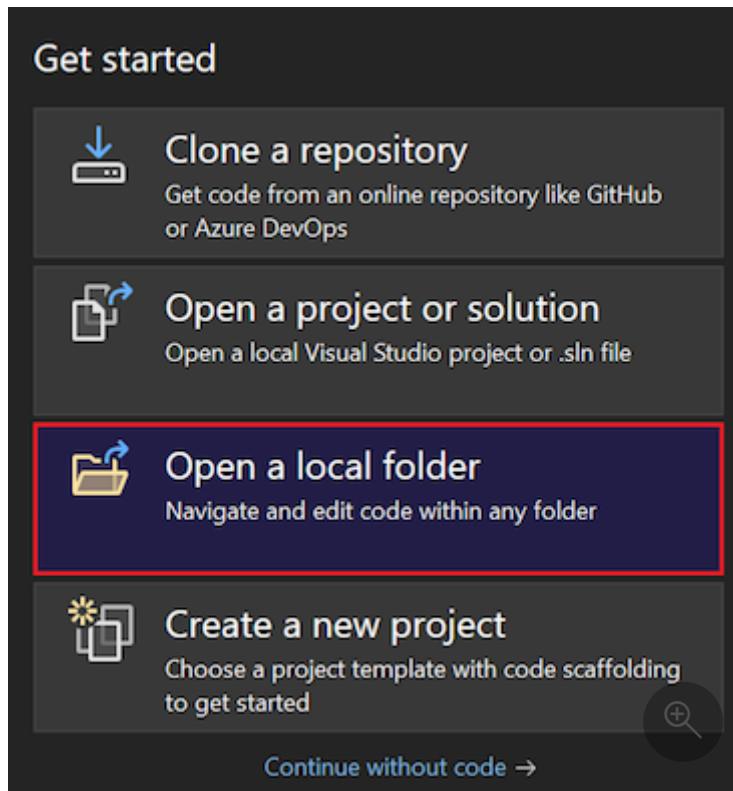
```
git clone https://github.com/gregmalcolm/python_koans
```

Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

Abra a pasta do código local

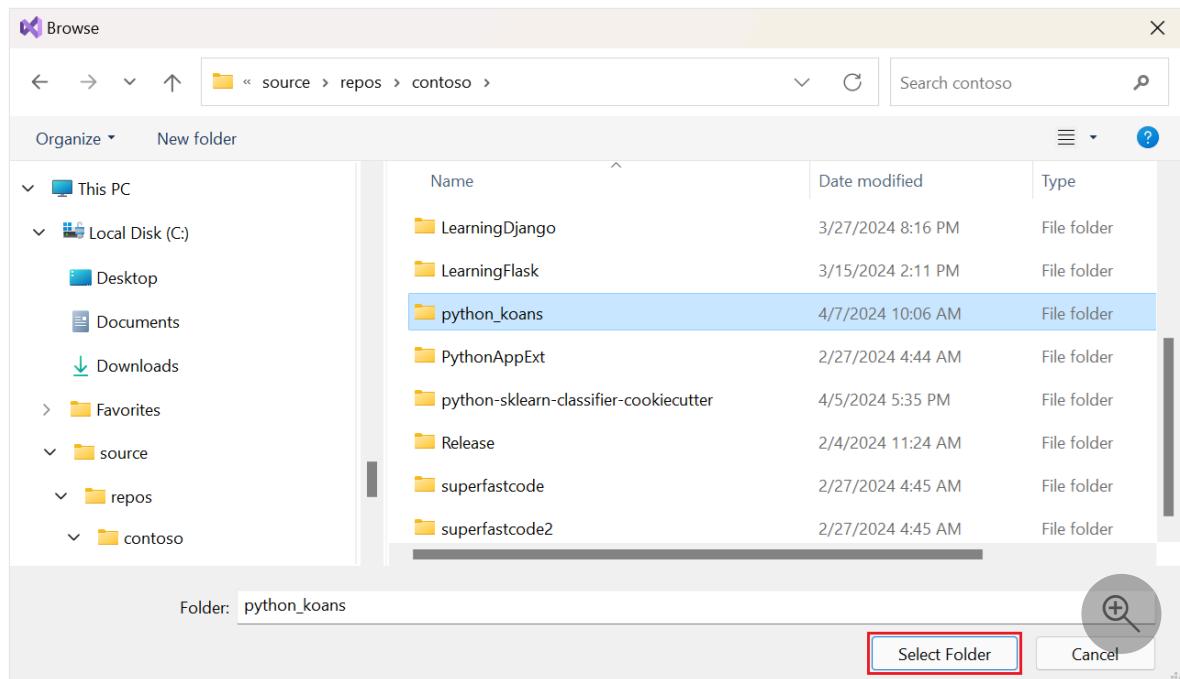
Siga estas etapas para abrir uma pasta local com o código do Python existente no Visual Studio:

1. Inicie o Visual Studio. Na janela Iniciar, selecione **Abrir uma pasta local** na coluna **Introdução**:

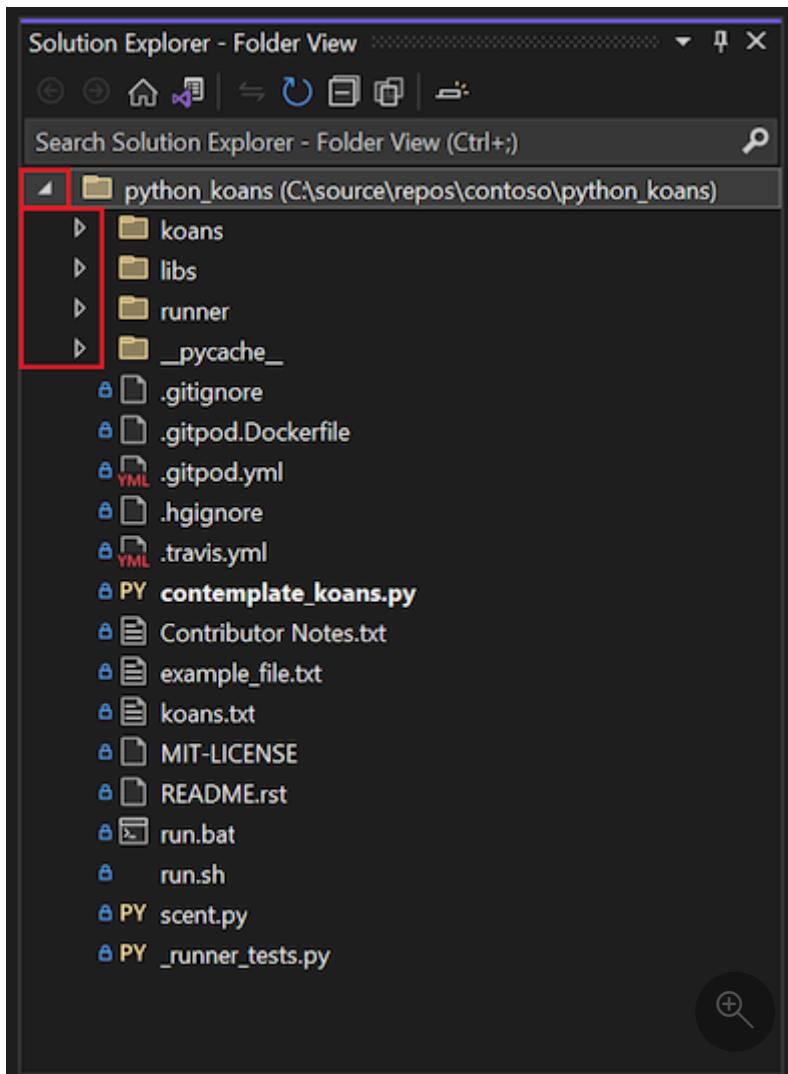


Se o Visual Studio já estiver em execução, selecione Arquivo>Abrir>Pasta.

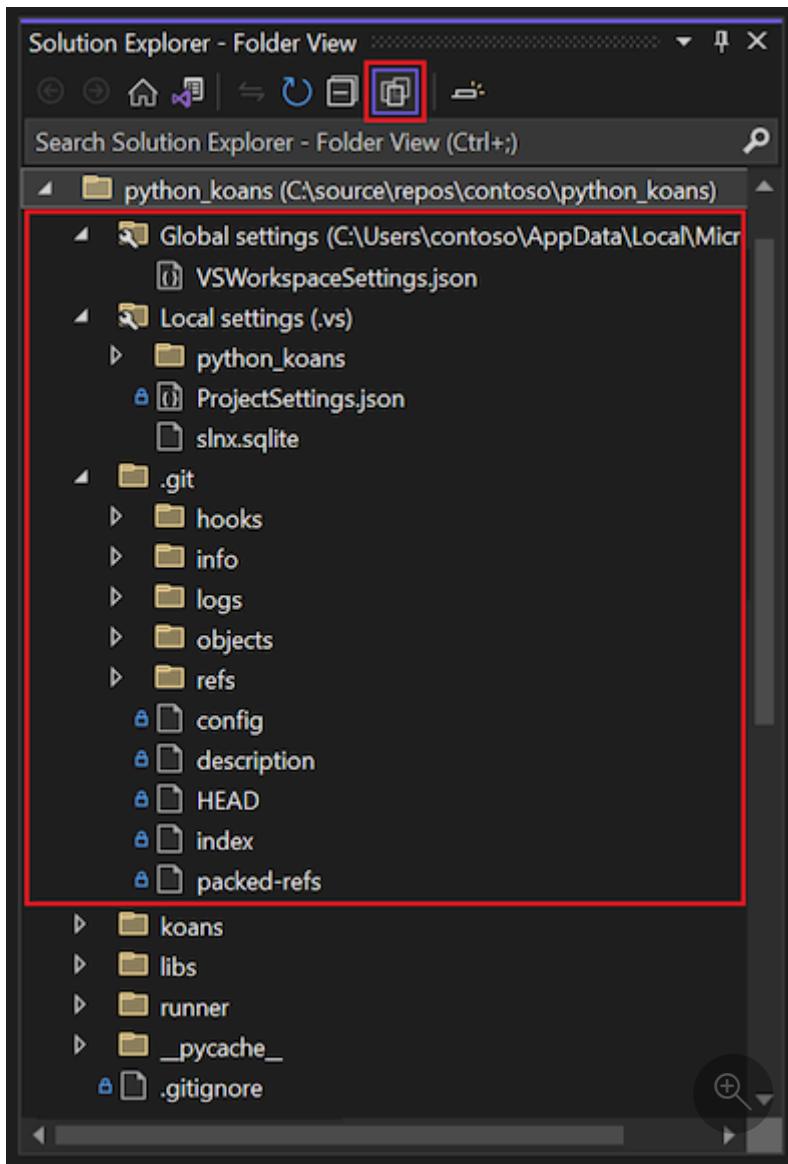
2. Navegue até a pasta que contém o código do Python e escolha **Selecionar Pasta**:



3. O Visual Studio exibe os arquivos no **Gerenciador de Soluções** na **Exibição de Pasta**. Você pode expandir e recolher uma pasta usando a seta à esquerda do nome da pasta:



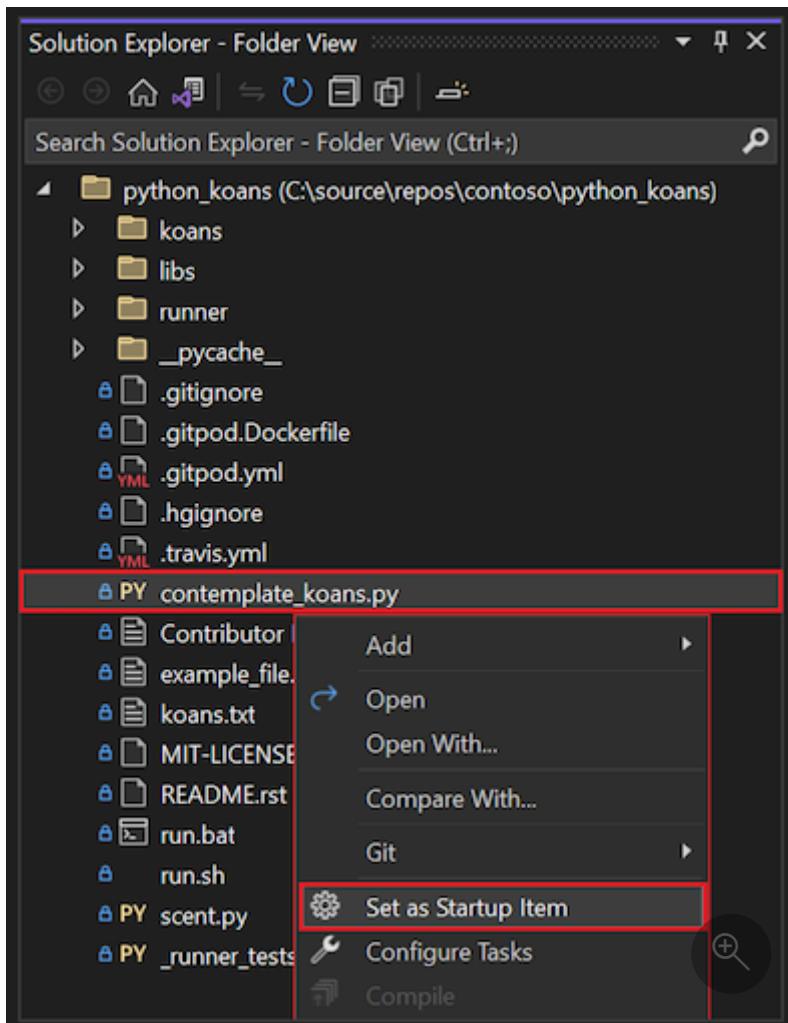
4. Ao abrir uma pasta do Python, o Visual Studio cria várias pastas ocultas para gerenciar as configurações relacionadas ao programa. Para ver essas pastas (e quaisquer outros arquivos e pastas ocultos, como a pasta `.git`), selecione o botão da barra de ferramentas **Mostrar todos os arquivos**:



Execute o programa

Depois de abrir o código Python existente no Visual Studio, você pode executar o programa. Para executar o código, você precisa identificar o **arquivo de inicialização** (item de inicialização) ou arquivo de programa primário para o Visual Studio executar o programa. Neste exemplo, o arquivo de inicialização é *contemplate-koans.py*.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no arquivo *contemplate-koans.py* e selecione **Definir como item de inicialização**:

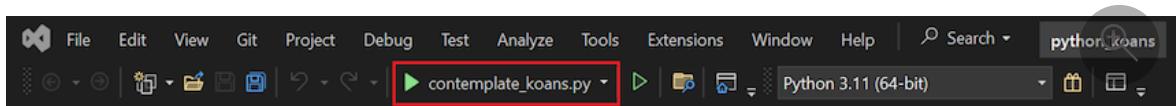


ⓘ Importante

Se o item de inicialização não estiver localizado na raiz da pasta que você abriu, você também deverá adicionar uma linha para o arquivo JSON de configuração de inicialização conforme descrito em [Definir um diretório de trabalho](#).

2. Execute o código selecionando **Depurar>Iniciar sem Depuração** ou use o atalho de teclado **Ctrl+F5**. Você também pode selecionar a seta sólida de reprodução ao lado do nome do item de inicialização na barra de ferramentas do Visual Studio. Esta opção executa o código no **Depurador** do Visual Studio.

Em todos esses métodos de inicialização, o Visual Studio detecta que seu item de inicialização é um arquivo Python e executa automaticamente o código no ambiente Python padrão. O ambiente atual é mostrado à direita do nome do item de inicialização na barra de ferramentas. No exemplo a seguir, o ambiente atual é **Python 3.11 (64 bits)**:



Se você não vir o **Ambiente Python** atual na barra de ferramentas, selecione **Exibir>Outras janelas>Ambientes Python**.

3. Quando o programa é executado, o Visual Studio abre uma janela de comando para exibir a saída do programa:

```
C:\WINDOWS\system32\cmd.exe
Thinking AboutAsserts
test_assert_truth has damaged your karma.

You have not yet reached enlightenment ...
AssertionError: False is not true

Please meditate on the following code:
File "C:\Python\gregmalcolm\python_koans\koans\about_asserts.py", line 17, in test_assert_truth
    self.assertTrue(False) # This should be True

You have completed 0 (0 %) koans and 0 (out of 37) lessons.
You are now 304 koans and 37 lessons away from reaching enlightenment.

Beautiful is better than ugly.
Press any key to continue . . .
```

➊ Observação

Se você executar o programa `python-koans` com depuração, será necessário alterar os valores no código para que o programa conclua a execução.

4. Você pode executar o código em um ambiente Python diferente:
 - a. Expanda a lista suspensa do **Ambiente Python** atual na barra de ferramentas do Visual Studio e selecione o ambiente desejado.
 - b. Reinicie o programa.
5. Quando estiver pronto para fechar a pasta no Visual Studio, selecione **Arquivo>Fechar pasta**.

Definir diretório de trabalho

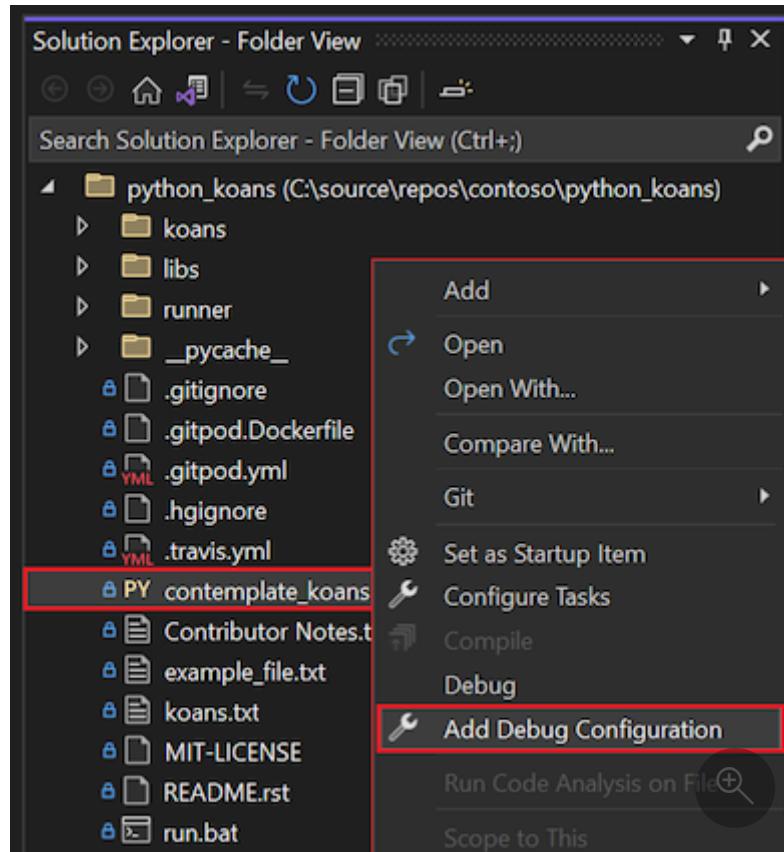
Por padrão, o Visual Studio executa um projeto do Python aberto como uma pasta na raiz da mesma pasta. No entanto, o código em seu projeto pode pressupor que o Python esteja sendo executado em uma subpasta. Quando seu código espera localizar arquivos em locais diferentes da configuração padrão reconhecida pelo Visual Studio, você talvez enfrente erros ao tentar executá-lo.

Suponha que você abra a pasta raiz do repositório `python_koans` e veja uma subpasta chamada `python3` que contém um arquivo do Python chamado `contemplate-koans.py`. Você decide definir o arquivo `python3/contemplate-koans.py` como **Arquivo de**

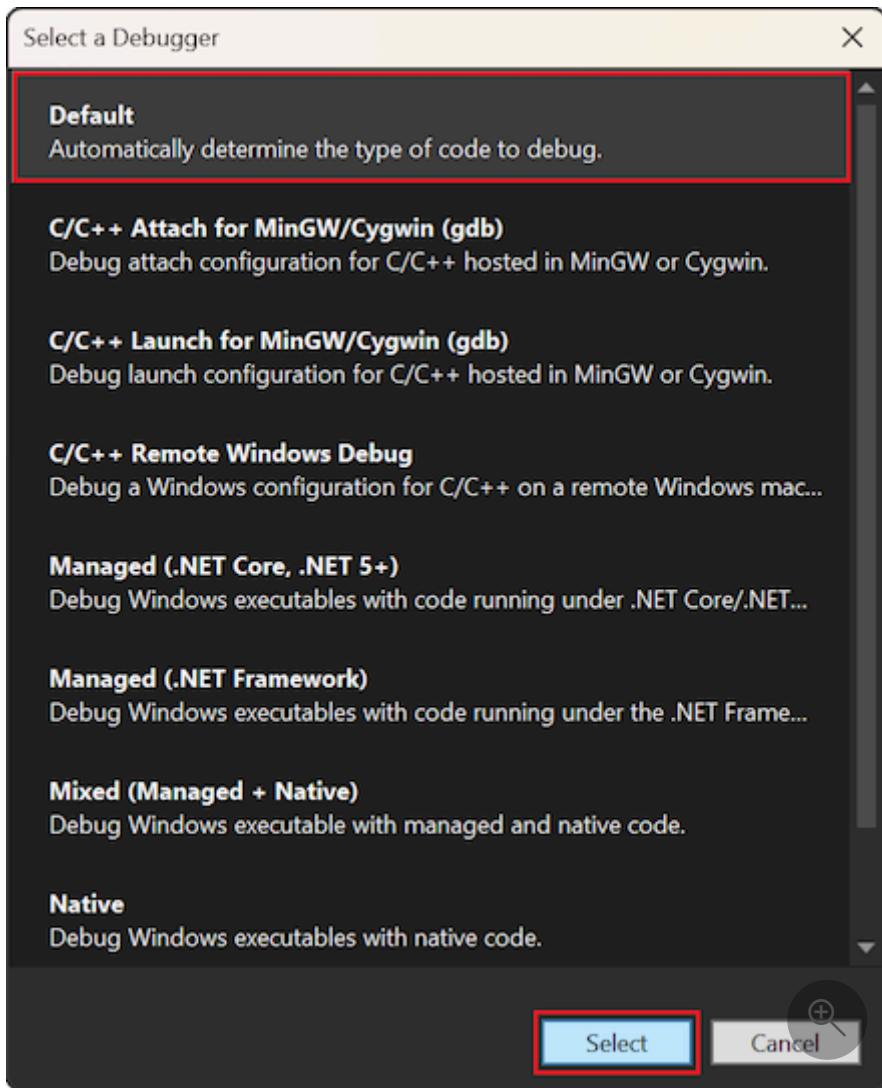
inicialização. Quando você executa o código, verá um erro informando que o arquivo denominado *koans.txt* não foi encontrado. O erro ocorre porque o arquivo *contemplate-koans.py* espera que o Python seja executado na pasta *python3* em vez de na raiz do repositório.

Nesses casos, você também deve adicionar uma linha ao arquivo JSON de configuração de inicialização para especificar o diretório de trabalho:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no arquivo de inicialização do Python (*.py*) e selecione **Adicionar Configuração de Depuração**:



2. Na caixa de diálogo, **Selecionar Depurador**, selecione a opção **Padrão** na lista e escolha **Selecionar**:



⚠ Observação

Se você não vir a opção **Padrão**, verifique se escolheu um arquivo `.py` do Python ao selecionar o comando **Adicionar Configuração de Depuração**. O Visual Studio usa o tipo de arquivo para determinar as opções do depurador para exibição.

- O Visual Studio abre um arquivo chamado `launch.json`, que está localizado na pasta `.vs` oculta. Este arquivo descreve o contexto de depuração para o projeto. Para especificar um diretório de trabalho, adicione um valor para a propriedade `"workingDirectory"`. Para o exemplo `python-koans`, você pode adicionar a propriedade e valor, `"workingDirectory": "python3"`:

JSON

```
{  
  "version": "0.2.1",  
  "defaults": {},  
  "configurations": [
```

```
{  
    "type": "python",  
    "interpreter": "(default)",  
    "interpreterArguments": "",  
    "scriptArguments": "",  
    "env": {},  
    "nativeDebug": false,  
    "webBrowserUrl": "",  
    "project": "contemplate_koans.py",  
    "projectTarget": "",  
    "name": "contemplate_koans.py",  
    "workingDirectory": "python3"  
}  
]  
}
```

4. Salve as alterações no arquivo *launch.json*.
5. Executar o programa novamente. A execução do código agora deve ser feita na pasta especificada.

Conteúdo relacionado

- Início Rápido: criar um projeto do Python com base em código existente
- Início Rápido: criar um projeto do Python de um repositório
- Tutorial: trabalhar com Python no Visual Studio

Comentários

Esta página foi útil?

 Yes

 No

Início Rápido: criar projeto do Python com base em um modelo no Visual Studio

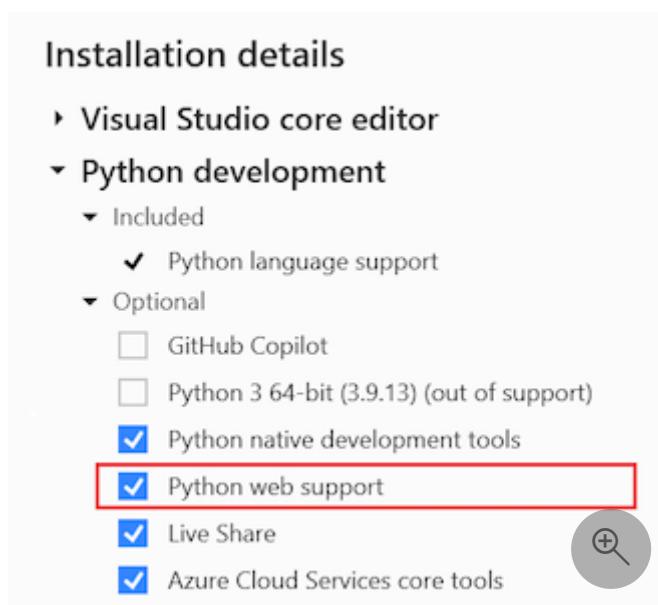
Artigo • 18/04/2024

Neste início rápido, você seguirá as etapas guiadas para rapidamente criar um aplicativo Flask usando um modelo interno de projeto Python. O Visual Studio facilita o desenvolvimento de projetos Python com código clichê e funcionalidade em modelos para várias estruturas Python, incluindo Flask, Django e Bottle. O projeto Python descrito neste artigo é semelhante ao projeto que você cria manualmente no artigo [Início Rápido: criar um aplicativo Web com o Flask](#).

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python.

No instalador do Visual Studio, selecione a carga de trabalho de **desenvolvimento do Python** e a opção de **suporte Web do Python** para acessar modelos de projeto da Web. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

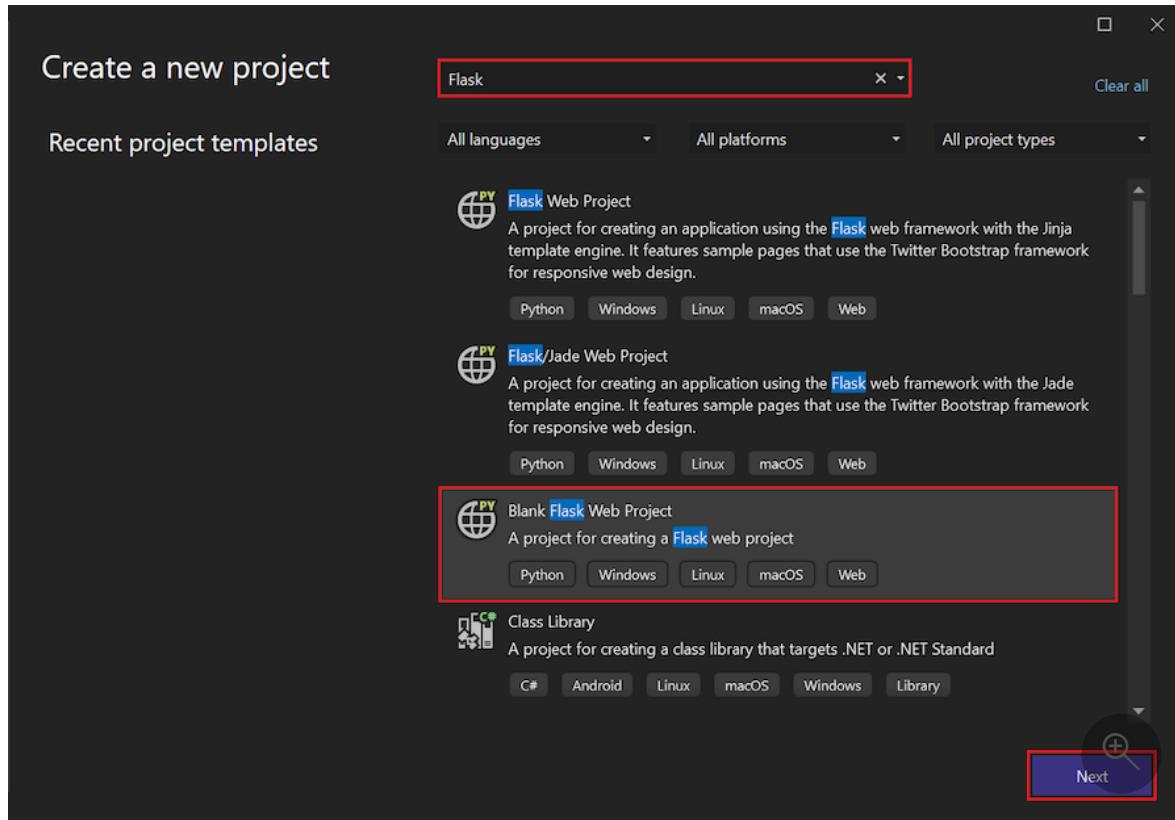


Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

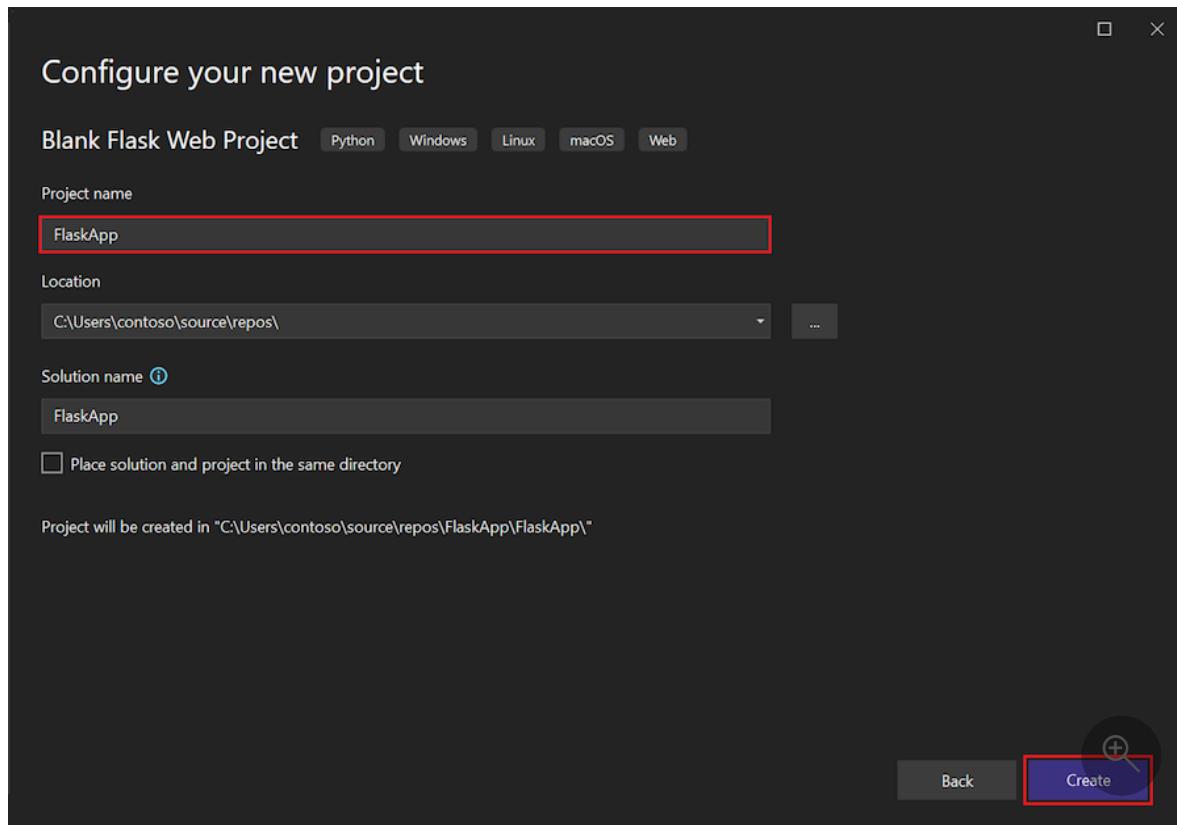
Criar o projeto

Siga este procedimento para criar uma solução do Visual Studio e um novo projeto Web do Flask:

1. No Visual Studio, escolha **Arquivo > Novo > Projeto**, pesquise "Flask" e selecione o modelo **Projeto Web do Flask em Branco** e, em seguida, selecione **Avançar**.



2. O Visual Studio exibe a caixa de diálogo de configuração do projeto. Insira um **Nome** para o projeto e selecione **Criar**. Você pode manter outros campos com a configuração padrão.



3. Depois de um momento, o Visual Studio exibe um prompt sobre como lidar com dependências de pacote. O prompt é diferente dependendo se você já tem dependências de pacote instaladas.

A caixa de diálogo **Este projeto requer pacotes externos**. indica que o modelo selecionado inclui um arquivo *requirements.txt* que especifica dependências no pacote Flask. O Visual Studio pode instalar os pacotes automaticamente, o que oferece a opção de instalá-los em um *ambiente virtual*. É recomendável usar um ambiente virtual na instalação em um ambiente global.

Se essa caixa de diálogo for exibida, selecione a opção **Instalar em um ambiente virtual**:

This project requires external packages

We can download and install these packages for you automatically, but we need to know whether you want to install them for just this project or for all your projects.

→ [Install into a virtual environment...](#)

Packages will only be available for this project (recommended)

→ [Install into Python 3.6 \(32-bit\)](#)

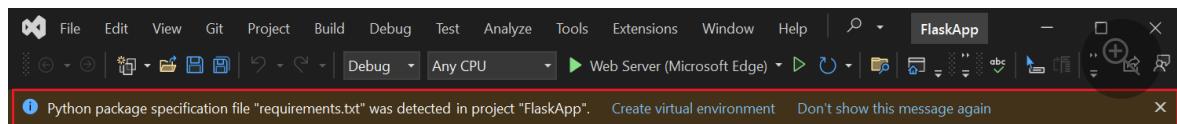
Packages will be shared by all projects

→ [I will install them myself](#)

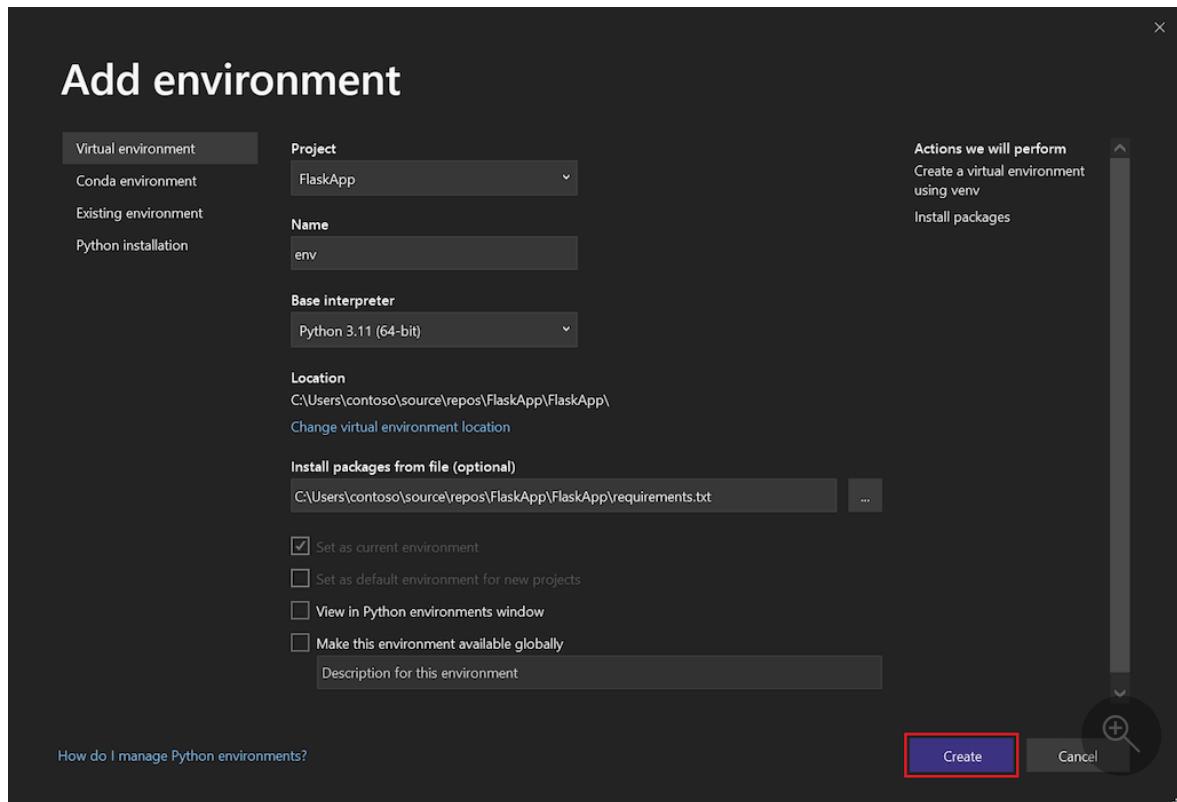


Como alternativa, você pode ver a solicitação arquivo de especificação do pacote Python "requirements.txt" foi detectado no projeto <Nome do projeto>. na parte superior da janela do Visual Studio. Essa solicitação indica que as dependências de pacote já estão disponíveis em sua configuração. O Visual Studio pode criar um ambiente virtual para você a partir da configuração existente.

Se você vir essa solicitação, selecione a opção Criar ambiente virtual:



4. O Visual Studio exibe a caixa de diálogo Adicionar ambiente. Aceite o valor padrão e selecione Criar, depois aprove todas as solicitações de elevação.



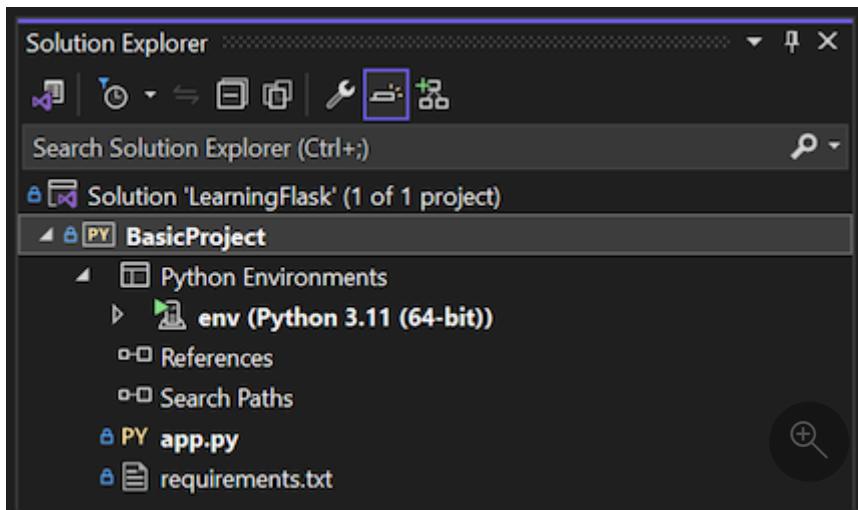
💡 Dica

Quando você inicia um projeto, é altamente recomendável criar um ambiente virtual imediatamente, como a maioria dos modelos do Visual Studio solicita. Ambientes virtuais mantêm os requisitos exatos do projeto ao longo do tempo, conforme você adiciona e remove bibliotecas. É possível gerar facilmente um arquivo *requirements.txt*, que você usa para reinstalar essas dependências em outros computadores de desenvolvimento (como quando usar controle do código-fonte) e ao implantar o projeto em um servidor de produção. Para saber mais sobre ambientes virtuais e seus benefícios, veja [Usar ambientes virtuais](#) e [Gerenciar os pacotes necessários com requirements.txt](#).

Examinar o código de texto clichê

Nesta seção, você examinará o código clichê no arquivo de projeto do Flask (*.py*) que o Visual Studio cria para o modelo selecionado.

1. Abra o **Gerenciador de Soluções** para exibir sua solução e os arquivos de projeto do Flask. O projeto inicial contém apenas dois arquivos, *app.py* e *requirements.txt*:



O arquivo *requirements.txt* especifica as dependências de pacote do Flask. A presença desse arquivo é que faz com que você seja convidado a criar um ambiente virtual ao desenvolver o projeto pela primeira vez.

O arquivo *app.py* único contém um código clichê para um projeto Web do Flask em branco. O modelo proporciona um código semelhante às descrições no artigo [Guia de início rápido - Criar um aplicativo Web com Flask](#), com a adição de algumas seções.

2. Abra o arquivo *app.py* no editor e examine a seção superior.

O código começa com uma instrução `import` para o pacote do Flask. Essa instrução cria uma instância da classe `Flask`, que é atribuída à variável `app`:

```
Python

from flask import Flask
app = Flask(__name__)
```

Em seguida, o código atribui a variável `wsgi_app`, que é útil quando você implanta o aplicativo Flask em um host da Web:

```
Python

# Make the WSGI interface available at the top level so wfastcgi can
# get it.
wsgi_app = app.wsgi_app
```

3. A seção do meio atribui uma função a uma rota de URL, o que significa que ela proporciona o recurso identificado pela URL. Neste caso, a rota define uma exibição:

Python

```
@app.route('/')
def hello():
    """Renders a sample page."""
    return "Hello World!"
```

Você defina rotas usando o decorador `@app.route` do Flask, cujo argumento é a URL relativa da raiz do site. Como é possível ver no código, a função retorna apenas uma cadeia de texto, que é suficiente para um navegador renderizar.

4. A seção inferior contém o código opcional para iniciar o servidor de desenvolvimento do Flask. Você pode definir o host e a porta por meio de variáveis de ambiente em vez de embuti-los em código. Esse código permite controlar facilmente a configuração nos computadores de desenvolvimento e produção sem alterar o código:

Python

```
if __name__ == '__main__':
    import os
    HOST = os.environ.get('SERVER_HOST', 'localhost')
    try:
        PORT = int(os.environ.get('SERVER_PORT', '5555'))
    except ValueError:
        PORT = 5555
    app.run(HOST, PORT)
```

5. Selecione **Depurar>Iniciar sem Depuração** para executar o aplicativo Flask e abrir um navegador para o host padrão e valor de porta, `localhost:5555`.

Explorar modelos de Python no Visual Studio

Quando você instala a carga de trabalho do Python, o Visual Studio proporciona vários modelos de projeto para as [estruturas Web do Flask, Bottle, Django](#) e serviços de nuvem do Azure. Há também modelos para diferentes cenários de aprendizado de máquina e um modelo para criar um projeto a partir de uma estrutura de pastas existente que contém um aplicativo Python. Você pode acessar os modelos usando a opção de menu **Arquivo>Novo>Projeto**. Selecione o nó da linguagem **Python** e seus nós filhos para ver os modelos disponíveis.

O Visual Studio também fornece uma variedade de arquivos ou *modelos de item* para criar rapidamente uma classe, pacote, teste de unidade do Python, arquivos `web.config` e muito mais. Quando houver um projeto do Python aberto, acesse os modelos de item

usando a opção de menu **Projeto > Adicionar novo item**. Para obter mais informações, veja a referência de [modelos de item](#).

Usar modelos pode economizar tempo significativo ao iniciar um projeto ou criar um arquivo. Eles também são uma ótima maneira de aprender sobre diferentes tipos de aplicativos e estruturas de código. É útil levar reservar alguns minutos para criar projetos e itens a partir de vários modelos para se familiarizar com o que eles oferecem.

Usar modelos do Cookiecutter

O Visual Studio proporciona a integração direta com o [Cookiecutter](#) para ajudar você a descobrir modelos e opções de modelo de entrada, além de criar projetos e arquivos. Para obter mais informações, consulte o artigo [Início Rápido: criar um projeto a partir de um modelo Cookiecutter](#).

Conteúdo relacionado

- [Tutorial: trabalhar com Python no Visual Studio](#)
- [Identificar manualmente um interpretador Python existente](#)
- [Diretório de instalação de ferramentas do Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Início Rápido: criar um projeto do Python com base em um código existente no Visual Studio

Artigo • 18/04/2024

Neste guia de início rápido, você segue as etapas guiadas para criar rapidamente um novo projeto do Python a partir do código existente. O Visual Studio facilita a transferência do código Python para um projeto do Visual Studio, com o assistente **Criar novo projeto a partir do código Python existente**.

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis ↗](#).

Use o assistente para criar um projeto com base em arquivos existentes

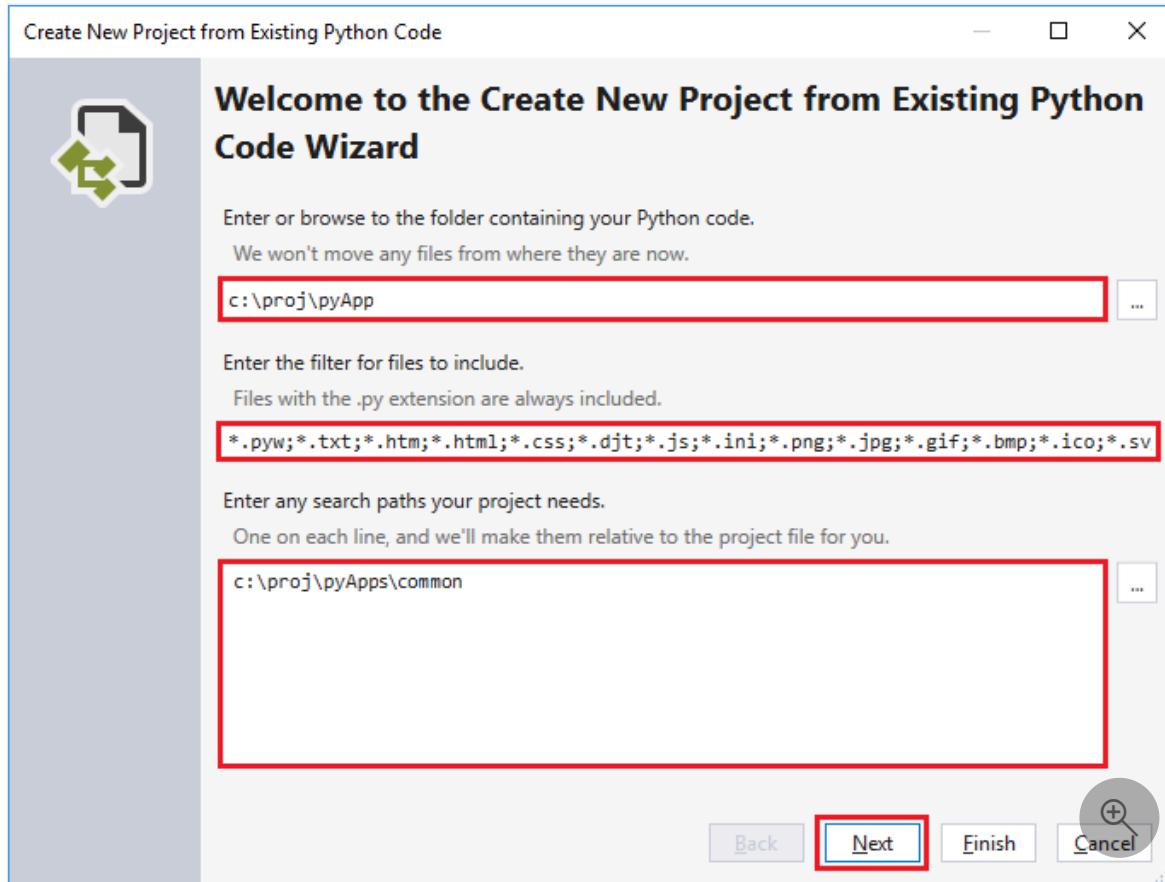
Siga estas etapas para criar um projeto com arquivos existentes.

ⓘ Importante

O processo a seguir não move nem copia nenhum arquivo de origem original. Se você quiser trabalhar com uma cópia de seus arquivos, primeiro duplique a pasta e, em seguida, crie o projeto.

1. Inicie o Visual Studio e selecione **Arquivo > Novo > Projeto**.
2. Na caixa de diálogo **Criar um projeto**, pesquise *python*, selecione o modelo **De código do Python existente** e **Avançar**.
3. Na caixa de diálogo **Configurar novo projeto**, insira um **Nome** e **Local** para o projeto, escolha a solução para contê-lo e selecione **Criar**.

4. No assistente Criar novo projeto do código Python existente, defina o **Caminho da pasta** para o código existente, defina um **Filtro** para tipos de arquivo e especifique os **Caminhos de pesquisa** necessários para o projeto e selecione **Avançar**. Se você não souber os caminhos de pesquisa, deixe o campo em branco.



5. Na próxima página, selecione o **Arquivo de inicialização** do projeto. O Visual Studio seleciona o interpretador e a versão globais padrão do Python. Você pode alterar o ambiente usando o menu suspenso. Quando estiver pronto, selecione **Próximo**.

ⓘ Observação

A caixa de diálogo mostra apenas arquivos na pasta raiz. Se o arquivo desejado estiver em uma subpasta, deixe o arquivo de inicialização em branco. Você pode definir o arquivo de inicialização no **Gerenciador de Soluções**, conforme descrito em uma etapa posterior.



Welcome to the Create New Project from Existing Python Code Wizard

Select the Python interpreter and version to use.

This setting can be changed later in Project Properties.

(Global default or an auto-detected virtual environment)

Choose which file to run when F5 is pressed.

If it is not in this list, you can right-click any file in your project and choose "Set as startup file"

module1.py

test1.py

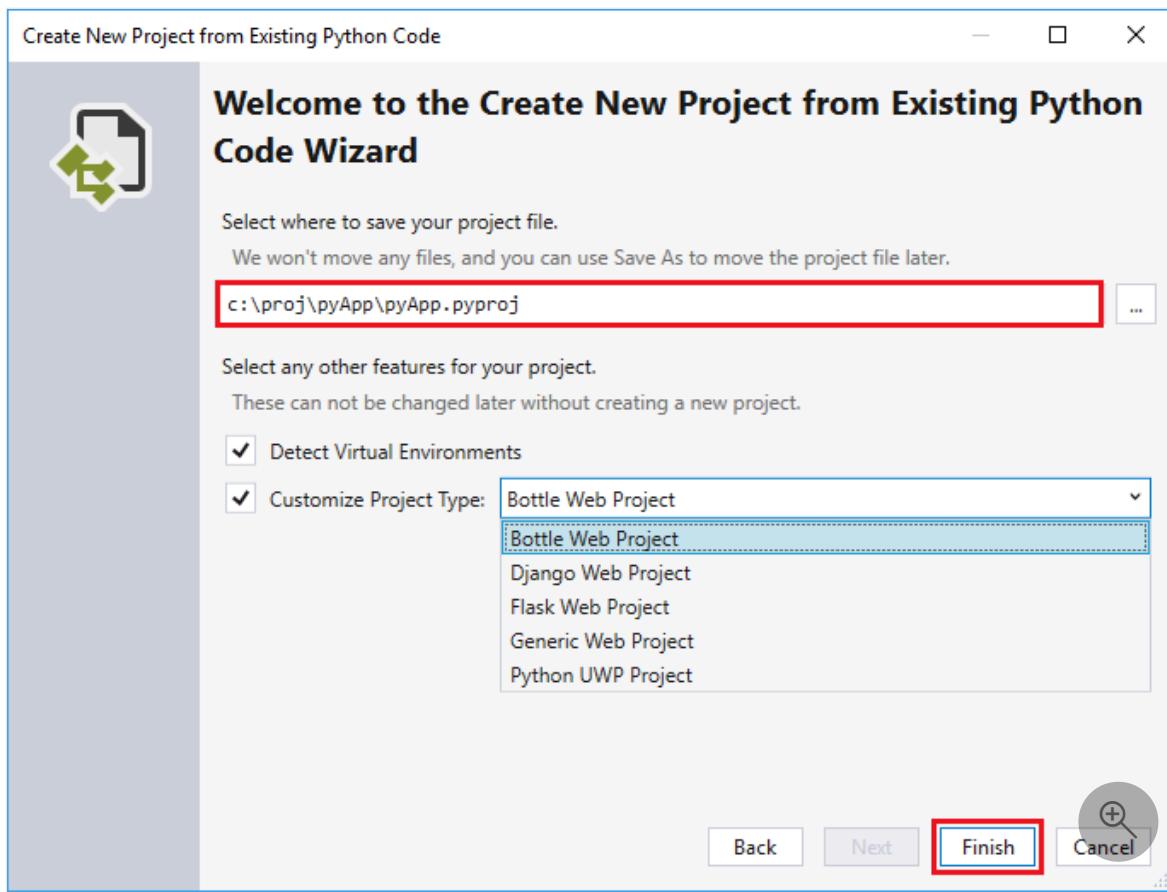
Back

Next

Finish



6. Selecione o local para armazenar o arquivo de projeto (um arquivo .pyproj no disco). Caso se aplique, também será possível incluir a detecção automática de ambientes virtuais e personalizar o projeto para outras estruturas da Web. Se você não tiver certeza sobre essas opções, deixe os campos com as configurações padrão.



7. Selecione Concluir.

O Visual Studio cria e abre o projeto no **Gerenciador de Soluções**. Se você quiser mover o arquivo `.pyproj` para um local diferente, selecione o arquivo no **Gerenciador de Soluções** e selecione Arquivo>Salvar como na barra de ferramentas. Essa ação atualizará as referências de arquivo no projeto, mas não moverá nenhum arquivo de código.

8. Para definir um arquivo de inicialização diferente, localize o arquivo no **Gerenciador de Soluções**, clique com o botão direito do mouse e selecione Definir como Arquivo de Inicialização.

Agora você pode executar seu programa selecionando Depurar>Iniciar sem Depuração na barra de ferramentas principal do Visual Studio ou usar o atalho do teclado Ctrl+F5.

Conteúdo relacionado

- [Tutorial: trabalhar com Python no Visual Studio](#)
- [Identificar manualmente um interpretador Python existente](#)
- [Início Rápido: abrir e executar o código do Python em uma pasta no Visual Studio](#)

Comentários

Esta página foi útil?

 Yes

 No

Início Rápido: clonar um repositório de código do Python no Visual Studio

Artigo • 18/04/2024

Neste guia de início rápido, você segue as etapas guiadas para clonar um repositório GitHub de um código do Python e criar um projeto. O Visual Studio facilita o trabalho com projetos do Python usando comandos Git para acessar o conteúdo sob o controle do código-fonte. Também é possível clonar repositórios do código do Python na linha de comando e, em seguida, trabalhar com os projetos no Visual Studio.

Pré-requisitos

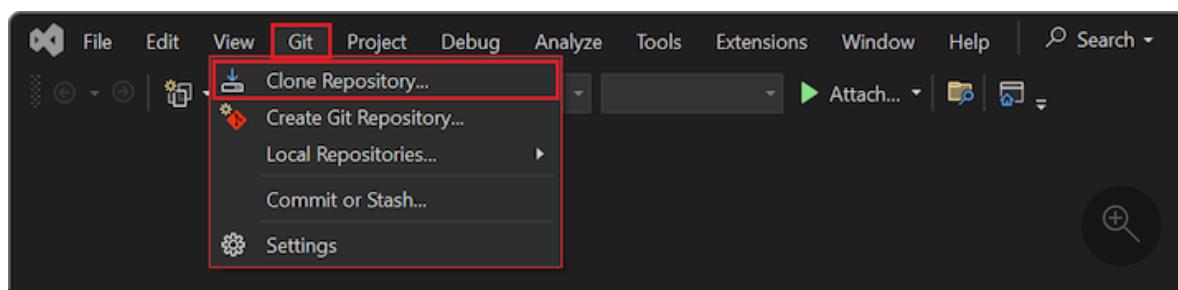
- Ter o Visual Studio 2022 instalado e compatível com cargas de trabalho do Python. O Visual Studio 2022 viabiliza a integração perfeita com o GitHub compatível com os comandos Git. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

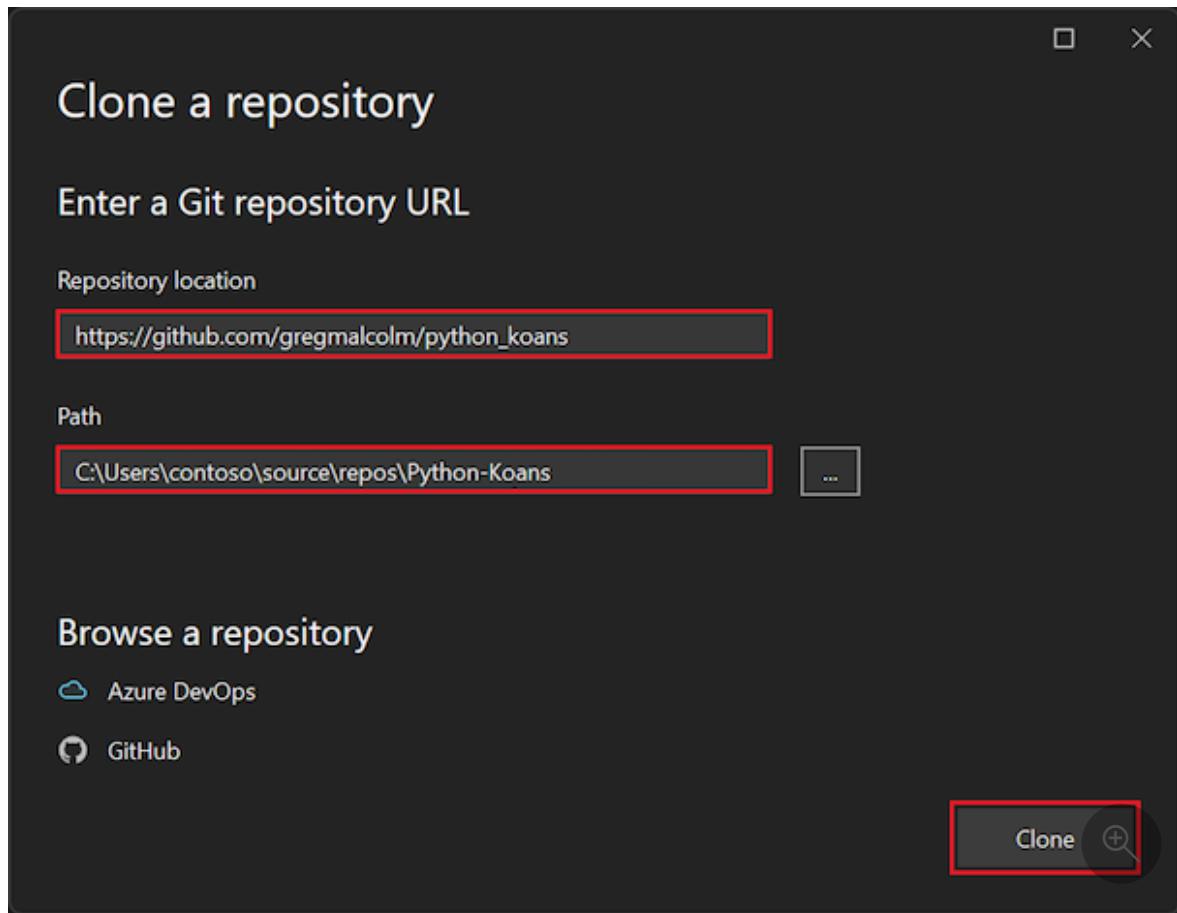
Clonar arquivos do repositório existente

Use as seguintes etapas para clonar um repositório existente usando os comandos de controle do código-fonte **Git** do Visual Studio:

1. No Visual Studio, selecione **Git>Clone**:



2. Na caixa de diálogo **Clonar um repositório**, especifique o repositório GitHub a ser clonado:



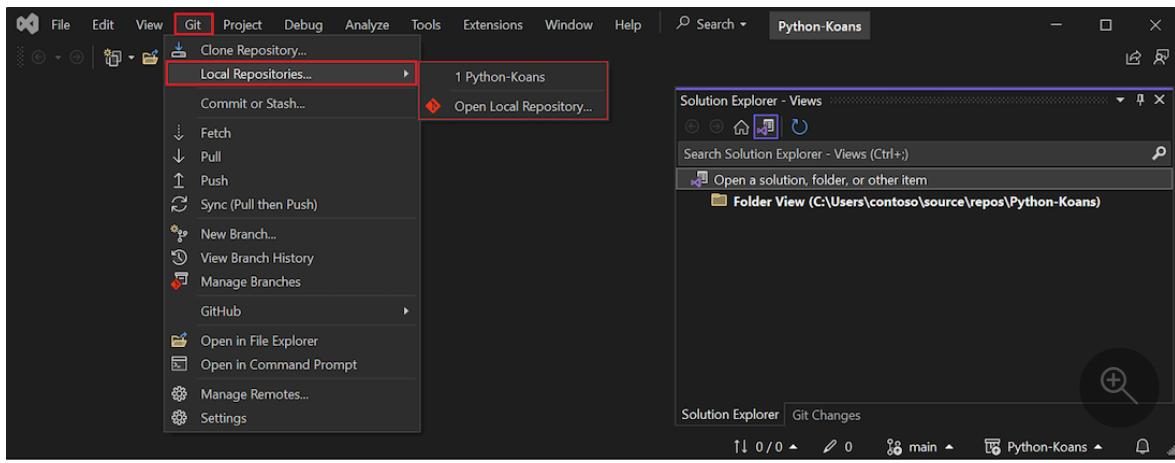
- No **local do repositório**, insira o URL do repositório que deve ser clonado. Para este exercício, insira `https://github.com/gregmalcolm/python_koans`.
- No **Caminho**, insira a pasta em seu sistema onde você deseja que o Visual Studio armazene os arquivos clonados.

A pasta especificada deve ser a pasta exata que você quer que o Visual Studio use. Ao contrário do comando `git clone`, ao criar um clone no **Team Explorer**, o Visual Studio não cria automaticamente uma subpasta com o nome do repositório.

- Quando estiver pronto, selecione **Clonar**.

Após a conclusão da clonagem, o Visual Studio abre o painel **Alterações do Git** para o repositório. Você também pode ver a pasta com os arquivos de repositório clonados no **Gerenciador de Soluções**.

- Selezione **Git>Repositórios Locais** e verifique se o repositório clonado está agora na lista:



ⓘ Importante

Neste guia de início rápido, você cria um clone direto do repositório *python_koans* no GitHub. Esse tipo de repositório é protegido pelo autor contra alterações diretas, portanto, a tentativa de confirmar as alterações no repositório falhará. Na prática, os desenvolvedores criam fork desse tipo de repositório em suas próprias contas do GitHub, fazem as alterações ali mesmo e, em seguida, criam solicitações de pull para enviar essas alterações para o repositório original. Quando você tiver seu próprio fork, use a URL dele em vez da URL do repositório original usada anteriormente.

Criar um projeto a partir de arquivos clonados

Depois de clonar o repositório, você pode criar um novo projeto a partir dos arquivos clonados.

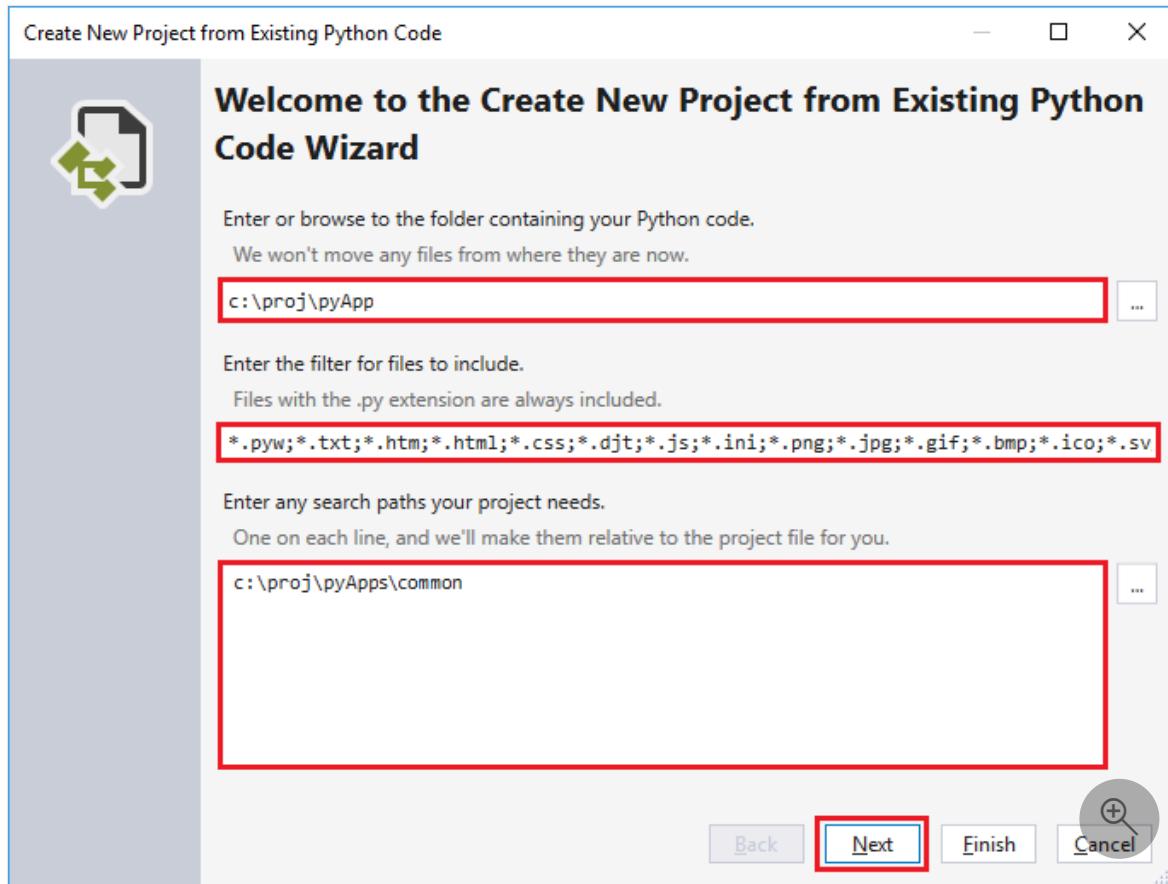
Siga estas etapas para criar um projeto com arquivos existentes.

ⓘ Importante

O processo a seguir não move nem copia nenhum arquivo de origem original. Se você quiser trabalhar com uma cópia de seus arquivos, primeiro duplique a pasta e, em seguida, crie o projeto.

1. Inicie o Visual Studio e selecione **Arquivo > Novo > Projeto**.
2. Na caixa de diálogo **Criar um projeto**, pesquise *python*, selecione o modelo **De código do Python existente** e **Avançar**.

3. Na caixa de diálogo **Configurar novo projeto**, insira um **Nome e Local** para o projeto, escolha a solução para contê-lo e selecione **Criar**.
4. No assistente **Criar novo projeto do código Python existente**, defina o **Caminho da pasta** para o código existente, defina um **Filtro** para tipos de arquivo e especifique os **Caminhos de pesquisa** necessários para o projeto e selecione **Avançar**. Se você não souber os caminhos de pesquisa, deixe o campo em branco.



5. Na próxima página, selecione o **Arquivo de inicialização** do projeto. O Visual Studio seleciona o interpretador e a versão globais padrão do Python. Você pode alterar o ambiente usando o menu suspenso. Quando estiver pronto, selecione **Próximo**.

ⓘ Observação

A caixa de diálogo mostra apenas arquivos na pasta raiz. Se o arquivo desejado estiver em uma subpasta, deixe o arquivo de inicialização em branco. Você pode definir o arquivo de inicialização no **Gerenciador de Soluções**, conforme descrito em uma etapa posterior.



Welcome to the Create New Project from Existing Python Code Wizard

Select the Python interpreter and version to use.

This setting can be changed later in Project Properties.

(Global default or an auto-detected virtual environment)

Choose which file to run when F5 is pressed.

If it is not in this list, you can right-click any file in your project and choose "Set as startup file"

module1.py

test1.py

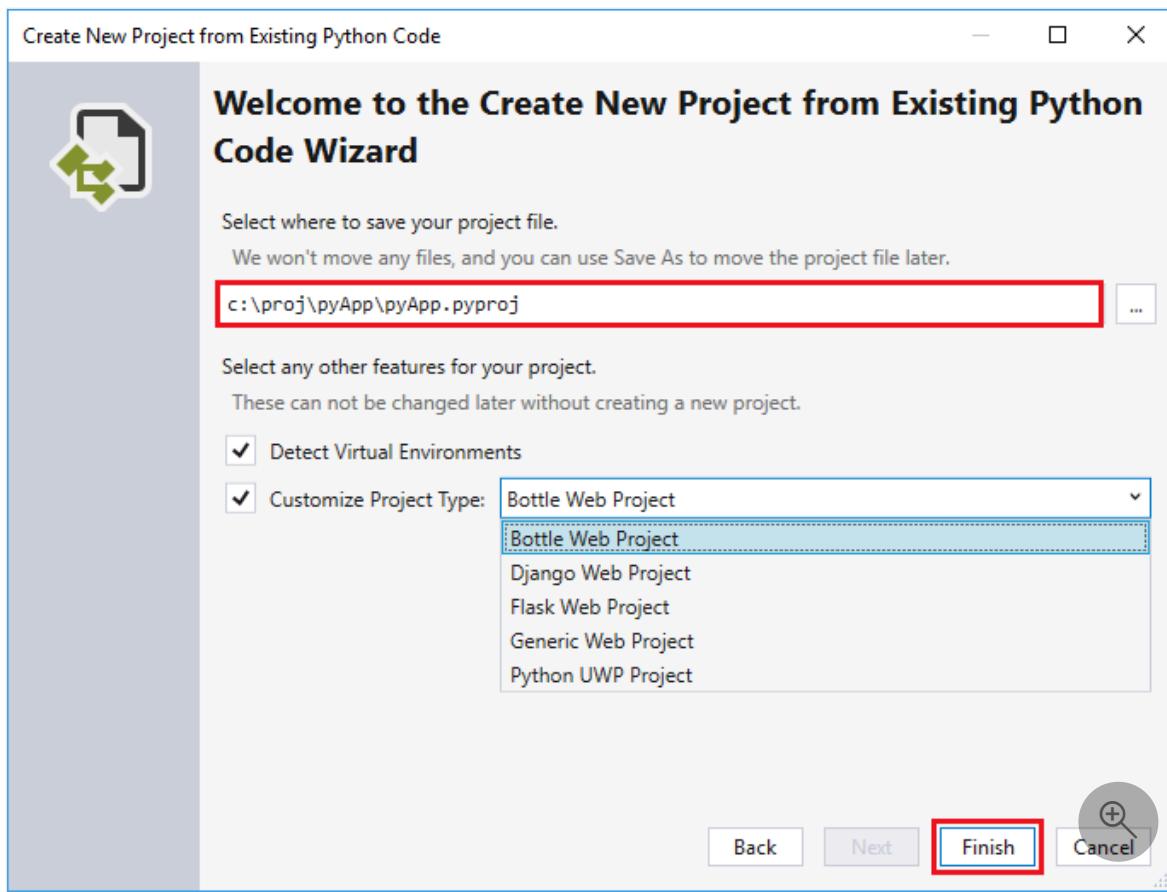
Back

Next

Finish



6. Selecione o local para armazenar o arquivo de projeto (um arquivo .pyproj no disco). Caso se aplique, também será possível incluir a detecção automática de ambientes virtuais e personalizar o projeto para outras estruturas da Web. Se você não tiver certeza sobre essas opções, deixe os campos com as configurações padrão.



7. Selecione Concluir.

O Visual Studio cria e abre o projeto no **Gerenciador de Soluções**. Se você quiser mover o arquivo `.pyproj` para um local diferente, selecione o arquivo no **Gerenciador de Soluções** e selecione **Arquivo > Salvar como** na barra de ferramentas. Essa ação atualizará as referências de arquivo no projeto, mas não moverá nenhum arquivo de código.

8. Para definir um arquivo de inicialização diferente, localize o arquivo no **Gerenciador de Soluções**, clique com o botão direito do mouse e selecione **Definir como Arquivo de Inicialização**.

Configurar as propriedades do projeto

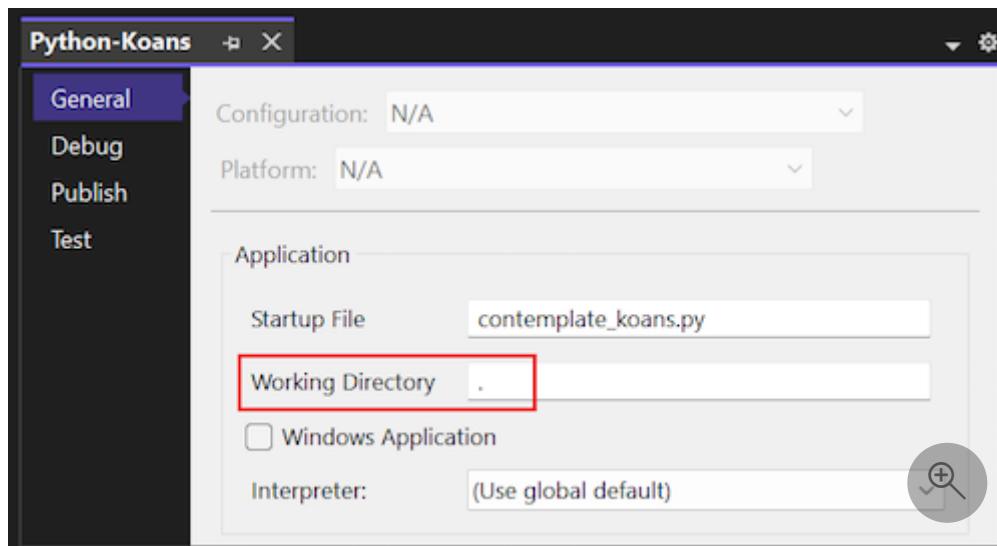
Para executar o projeto, você precisa identificar o diretório de trabalho para o projeto e informar ao Visual Studio qual arquivo usar como o **arquivo de inicialização**.

Siga estas etapas para configurar as propriedades do projeto:

1. No **Gerenciador de Soluções**, expanda o nó do projeto, clique com botão direito do mouse no arquivo `contemplate_koans.py` e selecione **Definir como arquivo de inicialização**. Essa ação permite que o Visual Studio saiba qual arquivo usar para executar o projeto.

2. Na barra de ferramentas principal do Visual Studio, selecione **Projeto>Propriedades** para abrir as respectivas Propriedades.
3. Na guia **Geral**, observe o valor do **Diretório de Trabalho** para o projeto.

Por padrão, o Visual Studio define o **Diretório de Trabalho** como a raiz do projeto (.). Observe que o **arquivo de inicialização** não tem nenhum local de pasta específico.

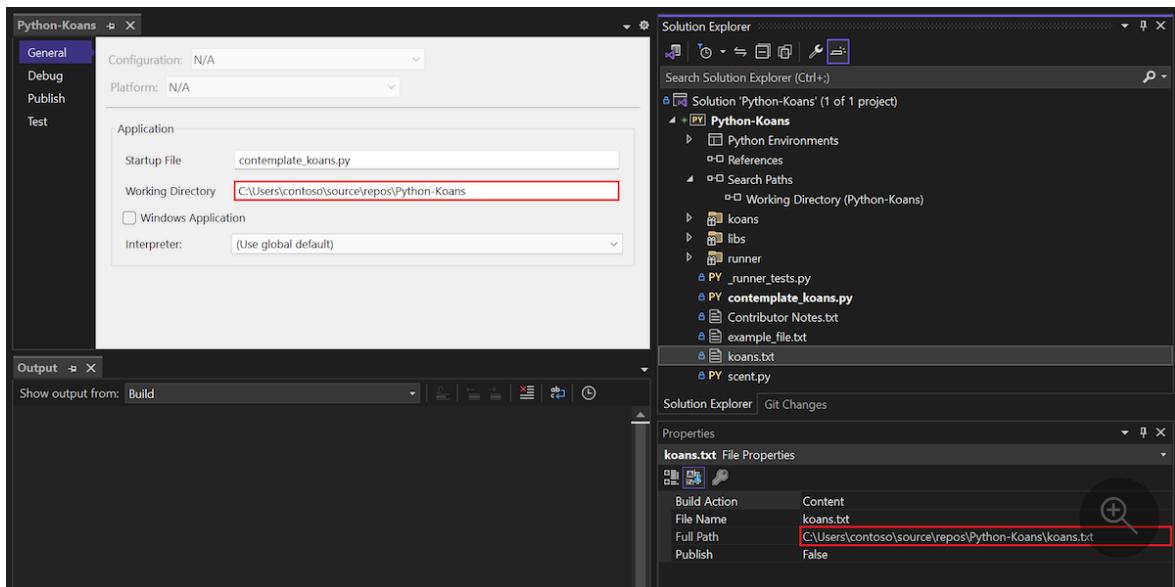


O código do programa clonado procura um arquivo chamado *koans.txt* no diretório de trabalho. O código espera que o diretório seja o local da pasta onde você instruiu o Visual Studio a armazenar os arquivos de repositório clonados. Se você deixar o **Diretório de Trabalho** definido como a raiz do projeto (.), o programa gerará erros de runtime.

4. Defina o valor do **Diretório de Trabalho** como o local da pasta do repositório clonado, como `C:\Users\contoso\source\repos\Python-Koans`.

Dica

Uma maneira rápida de confirmar o local da pasta para os arquivos clonados é verificar as propriedades do arquivo clonado no **Gerenciador de Soluções**. Clique com o botão direito do mouse no arquivo *koans.txt* e selecione **Propriedades** para abrir o painel Detalhes no **Gerenciador de Soluções**. No painel Detalhes, observe que o local da pasta para o arquivo está listado na propriedade **Caminho completo**. Você pode colar esse valor no campo **Diretório de Trabalho** na página **Propriedades** do projeto.



5. Salve as alterações e feche o painel Propriedades do projeto.

Executar programa do Python

Agora você está pronto para tentar executar o aplicativo para o novo projeto:

1. Selecione Depurar>Iniciar sem Depuração (ou use o atalho de teclado Ctrl+F5) para executar o programa.

Se você vir o erro de runtime **FileNotFoundException** para o arquivo *koans.txt*, verifique se o **Diretório de Trabalho** está definido corretamente, conforme descrito na [seção Anterior](#).

2. Quando o programa é executado com êxito, ele exibe um erro de asserção na linha 17 do arquivo de projeto */koans/about_asserts.py*:

```
C:\Windows\system32\cmd.exe
Thinking AboutAsserts
test_assert_truth has damaged your karma.

You have not yet reached enlightenment ...
AssertionError: False is not true

Please meditate on the following code:
  File "c:\repo\koans\python3\koans\about_asserts.py", line 17, in test_assert_truth
    self.assertTrue(False) # This should be True

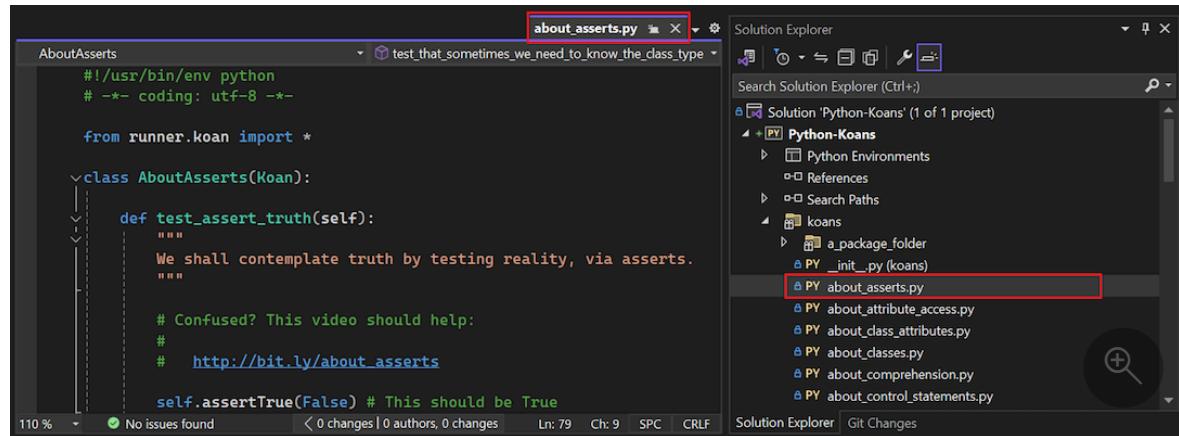
You have completed 0 (0 %) koans and 0 (out of 37) lessons.
You are now 302 koans and 37 lessons away from reaching enlightenment.

Beautiful is better than ugly.
Press any key to continue . . .
```

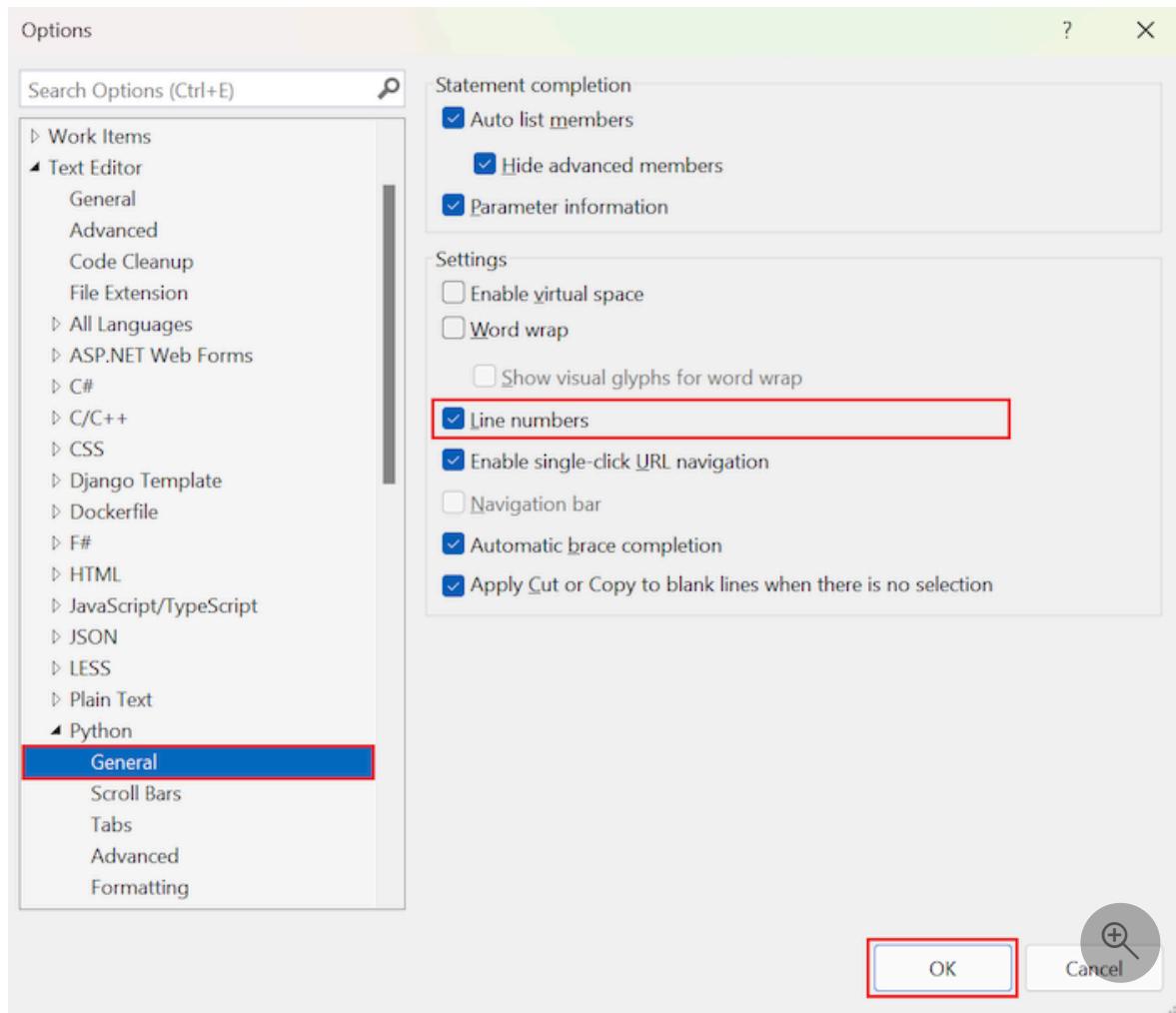
O erro de asserção é intencional. O programa foi projetado para ensinar Python, fazendo com que você corrija todos os erros intencionais. Você pode encontrar mais informações sobre o programa em [Ruby Koans](#), que serviu de inspiração para o Python Koans.

3. Saia do programa.

4. No Gerenciador de Soluções, dê um clique duplo no arquivo `/koans/about_asserts.py` para abrir o arquivo no editor:



Por padrão, números de linha não aparecem no editor. Para ver os números de linha, selecione **Ferramentas>Opções**, expanda a seção **Editor de Texto>Python>Geral** e selecione a opção **Números de linha**:



5. Corrija o erro no arquivo `/koans/about_asserts.py` alterando o argumento `False` na linha 17 para `True`. Esta deve ser a aparência do código atualizado:

```
Python

self.assertTrue(True) # This value should be True
```

6. Execute o programa novamente.

Se o Visual Studio alertar sobre erros, responde com **Sim** para continuar a execução do código. Desta vez, o programa passa pela primeira verificação e para no koan seguinte. Você pode continuar a corrigir outros erros e executar o programa para ver os ajustes.

Conteúdo relacionado

- [Clonar um repositório Git no Visual Studio](#)
- [Criar um repositório Git do Visual Studio](#)
- [Tutorial: trabalhar com Python no Visual Studio](#)

Comentários

Esta página foi útil?

 Yes

 No

Início Rápido: criar um projeto por meio de um modelo do Cookiecutter

Artigo • 18/04/2024

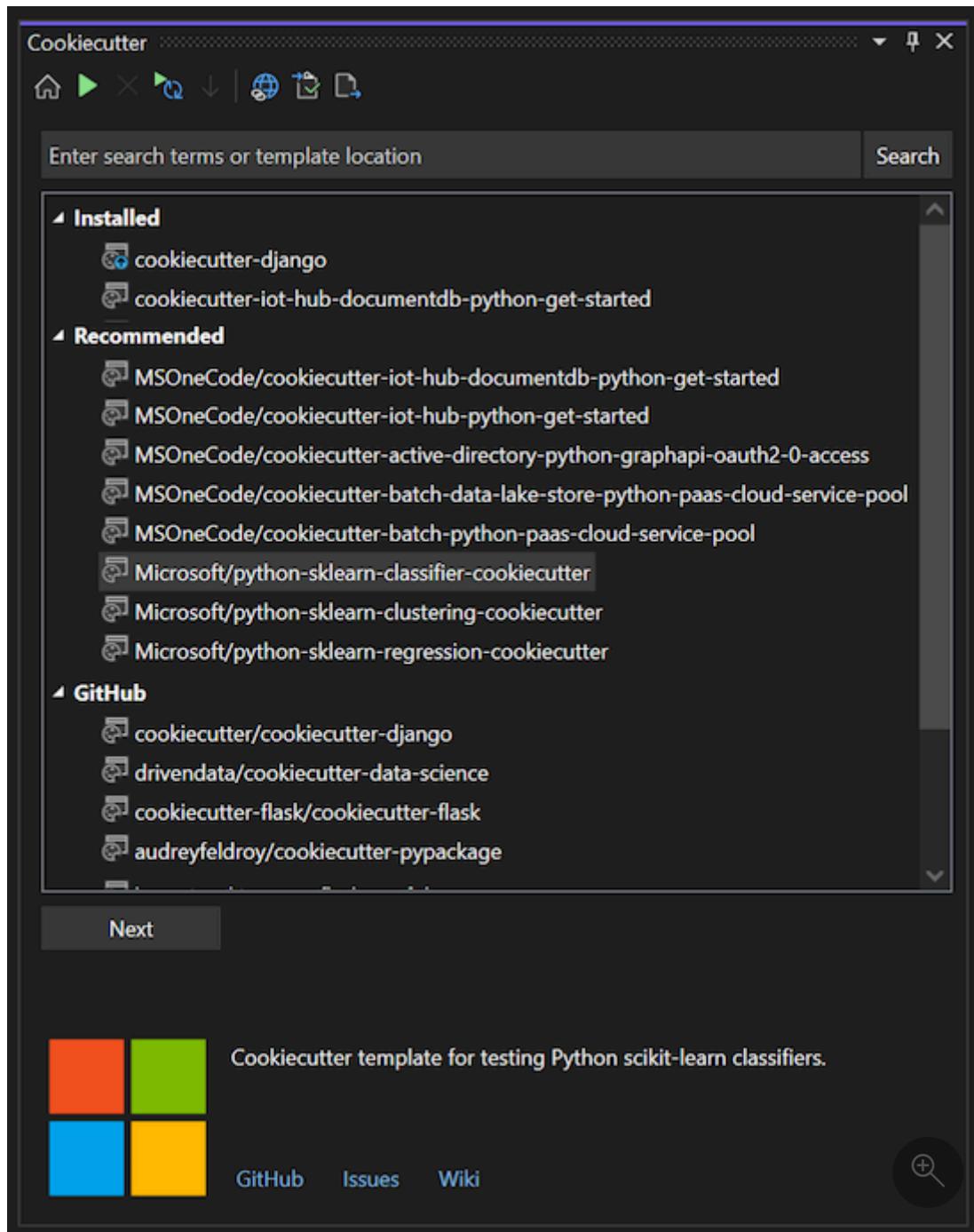
Neste guia de início rápido, você segue as etapas guiadas para criar um novo projeto usando um modelo do Cookiecutter, incluindo muitos que estão publicados no GitHub. O [Cookiecutter](#) fornece uma interface gráfica do usuário para descobrir modelos e opções de modelo de entrada e criar projetos e arquivos. O Visual Studio 2017 e posterior inclui a extensão Cookiecutter. Ela pode ser instalada separadamente em versões anteriores do Visual Studio.

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Python 3.3 ou posterior (32 ou 64 bits) ou o Anaconda 3 versão 4.2 ou posterior (32 ou 64 bits).
 - Se um interpretador do Python adequado não estiver disponível, o Visual Studio exibirá um aviso.
 - Se você instalar um interpretador do Python enquanto o Visual Studio estiver em execução, selecione a opção **Início** na barra de ferramentas do **Cookiecutter Explorer** para detectar o interpretador recém-instalado. Para obter mais informações, consulte [Criar e gerenciar ambientes do Python no Visual Studio](#).

Criar um projeto usando o Cookiecutter

1. No Visual Studio, selecione **Arquivo > Novo > A partir do Cookiecutter**. Esse comando abre a janela do **Cookiecutter** no Visual Studio, onde você pode procurar modelos.

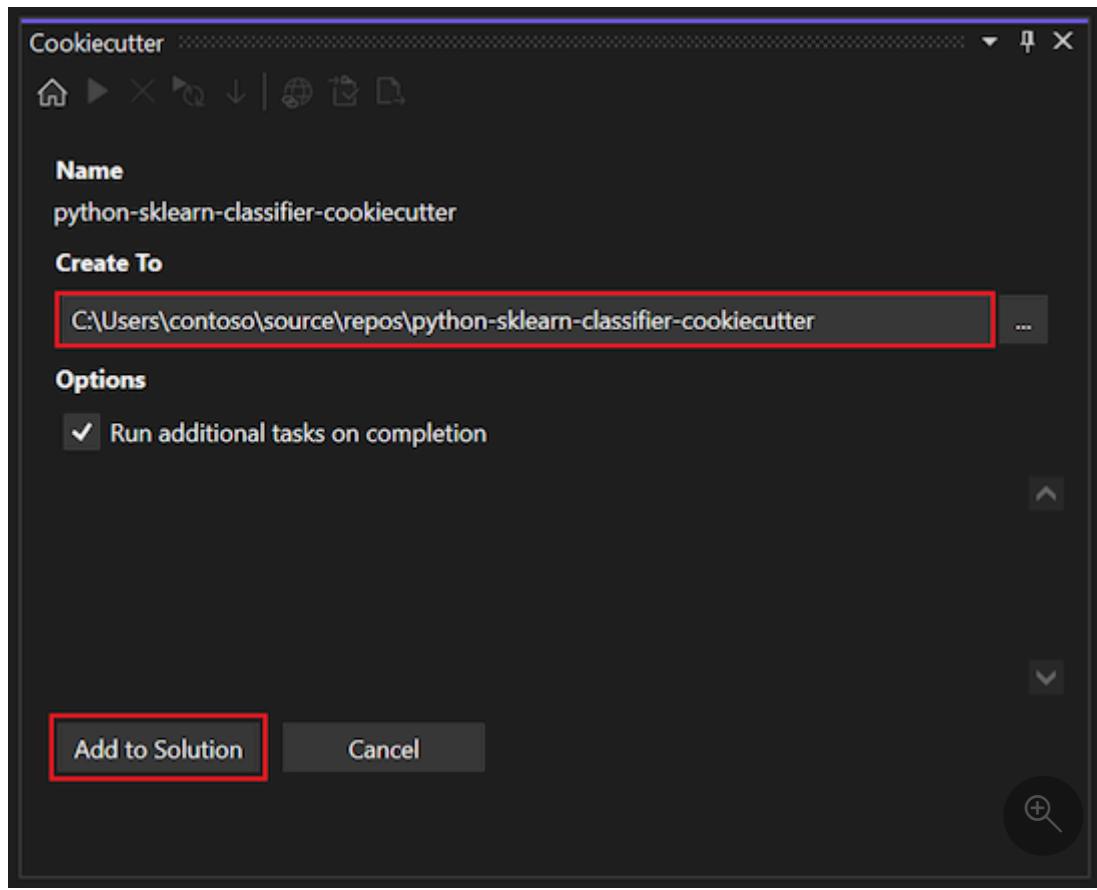


2. Na janela do **Cookiecutter**, selecione o modelo **Microsoft/python-sklearn-classifier-cookiecutter** na seção **Recomendado**.

3. Para clonar e instalar o modelo selecionado, selecione **Avançar**.

O processo poderá levar vários minutos na primeira vez que você usar um modelo específico, já que o Visual Studio instala os pacotes do Python necessários.

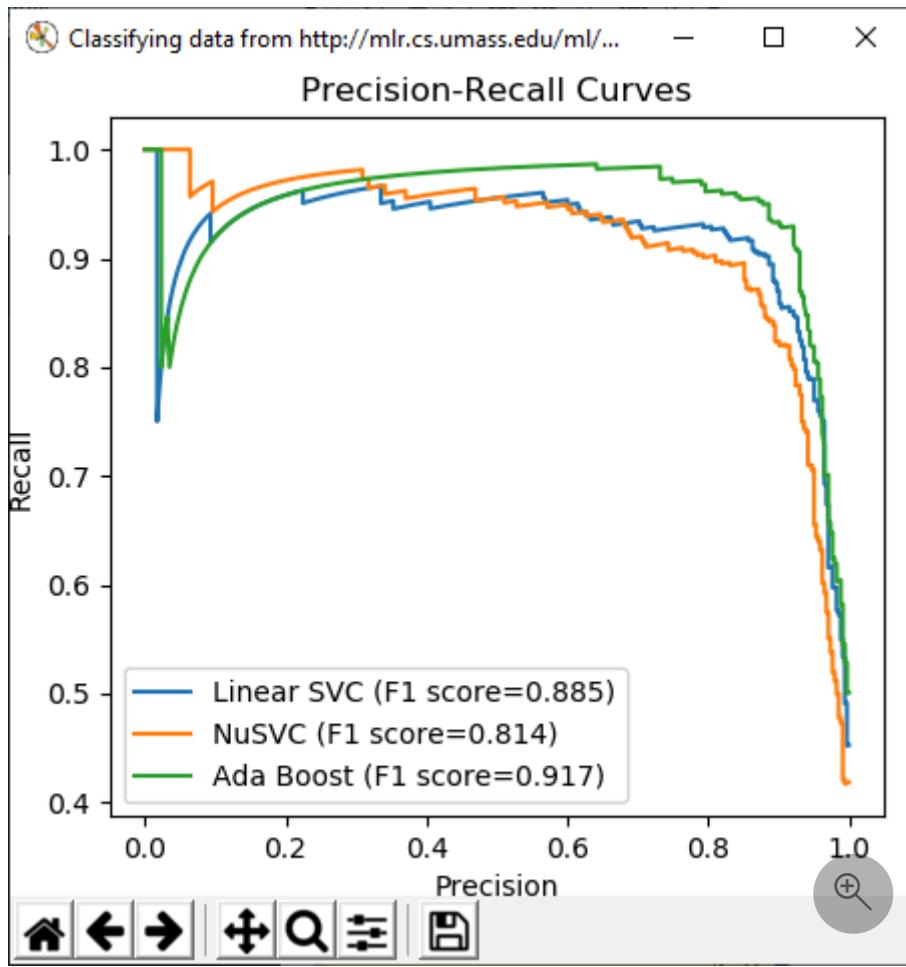
4. Depois de instalar e clonar um modelo localmente, o **Cookiecutter** exibe a página **Opções**, onde você especifica configurações para o seu novo projeto.



- a. Defina o campo **Criar para** como o local onde você quer que o Visual Studio armazene os novos arquivos de projeto, como `C:\repos\python-sklearn-classifier-cookiecutter\`.
- b. Selecione **Criar e abrir projeto**. (Se você estiver adicionando o novo projeto a uma solução existente, verá a opção **Adicionar à Solução**.)

Quando o processo for concluído, você verá a mensagem **Arquivos criados com êxito usando o modelo....** O projeto é aberto no **Gerenciador de Soluções** automaticamente.

5. Para executar o programa, selecione **Depurar>Iniciar sem Depuração** ou use o atalho de teclado **Ctrl+F5**. O programa gera um gráfico de curva de recall de precisão:



Conteúdo relacionado

- Usar a extensão Cookiecutter
- Tutorial: trabalhar com Python no Visual Studio

Comentários

Esta página foi útil?

Yes

No

Tutorial: Trabalhar com Python no Visual Studio

Artigo • 09/02/2024

Neste tutorial, você aprenderá a trabalhar com o Python no Visual Studio. O Python é uma linguagem de programação popular que é confiável, flexível, fácil de aprender de uso gratuito em todos os sistemas operacionais. O Python tem suporte de uma comunidade de desenvolvedores forte e conta com muitas bibliotecas gratuitas. O Python dá suporte a todos os tipos de desenvolvimento, incluindo aplicativos Web, serviços Web, aplicativos da área de trabalho, scripts e computação científica. Muitas universidades, cientistas, desenvolvedores casuais e desenvolvedores profissionais usam o Python. O Visual Studio fornece suporte de linguagem de primeira classe para o Python.

Este tutorial orienta você em um processo de seis etapas:

- ✓ [Etapa 1: criar um projeto do Python \(este artigo\)](#)
- ✓ [Etapa 2: escrever e executar código para ver o IntelliSense do Visual Studio funcionando](#)
- ✓ [Etapa 3: criar mais código na janela REPL interativa](#)
- ✓ [Etapa 4: executar o programa concluído no depurador do Visual Studio](#)
- ✓ [Etapa 5: instalar pacotes e gerenciar ambientes do Python](#)
- ✓ [Etapa 6: Trabalhar com o Git](#)

Este artigo aborda as tarefas na **Etapa 1**. Você cria um projeto e examina os elementos da interface do usuário visíveis no Gerenciador de Soluções.

Pré-requisitos

Visual Studio com a carga de trabalho do Python instalada. Para obter instruções, confira [Instalar as ferramentas do Python para Visual Studio](#).

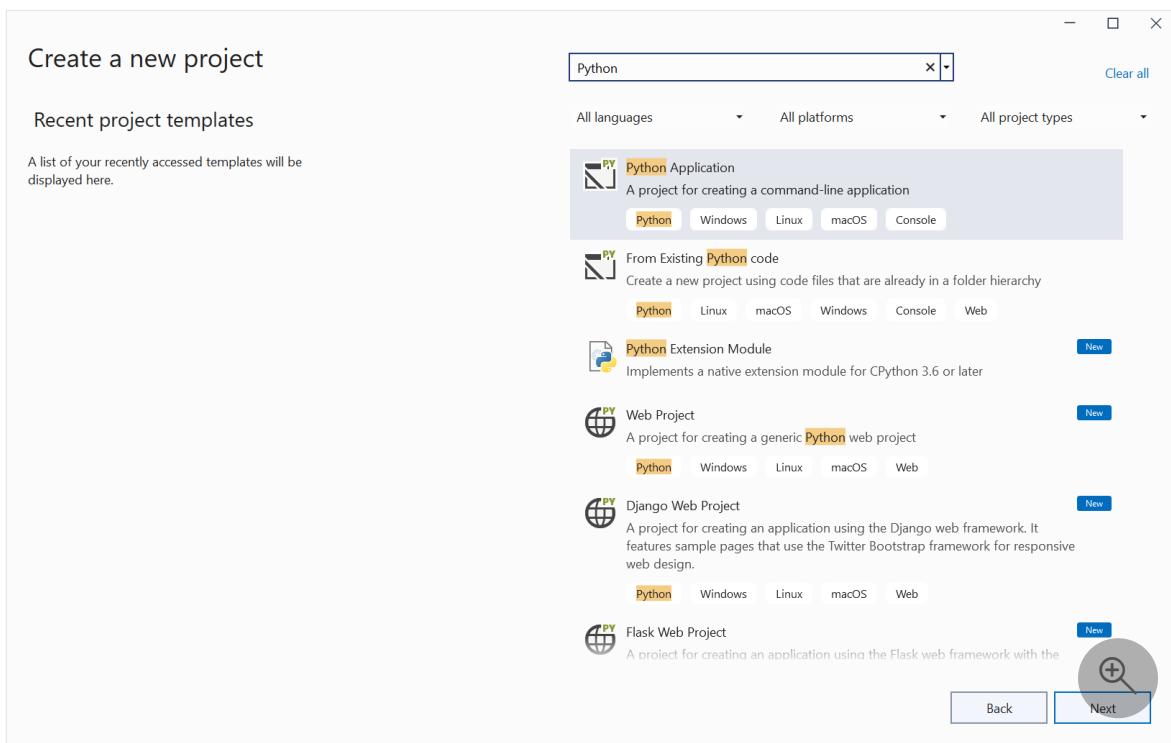
Etapa 1: criar um novo projeto do Python

Um *projeto* é como o Visual Studio gerencia todos os arquivos que se reúnem para produzir um determinado aplicativo. Os arquivos de aplicativo incluem código-fonte, recursos e configurações. Um projeto formaliza e mantém as relações entre todos os arquivos do projeto. O projeto também gerencia recursos externos que são compartilhados entre vários projetos. Um projeto permite que seu aplicativo expanda e

cresça sem esforço. Usar projetos é mais fácil do que gerenciar relacionamentos manualmente em pastas não planejadas, scripts, arquivos de texto e na sua memória.

Neste tutorial você começará com um projeto simples, contendo apenas um arquivo de código vazio.

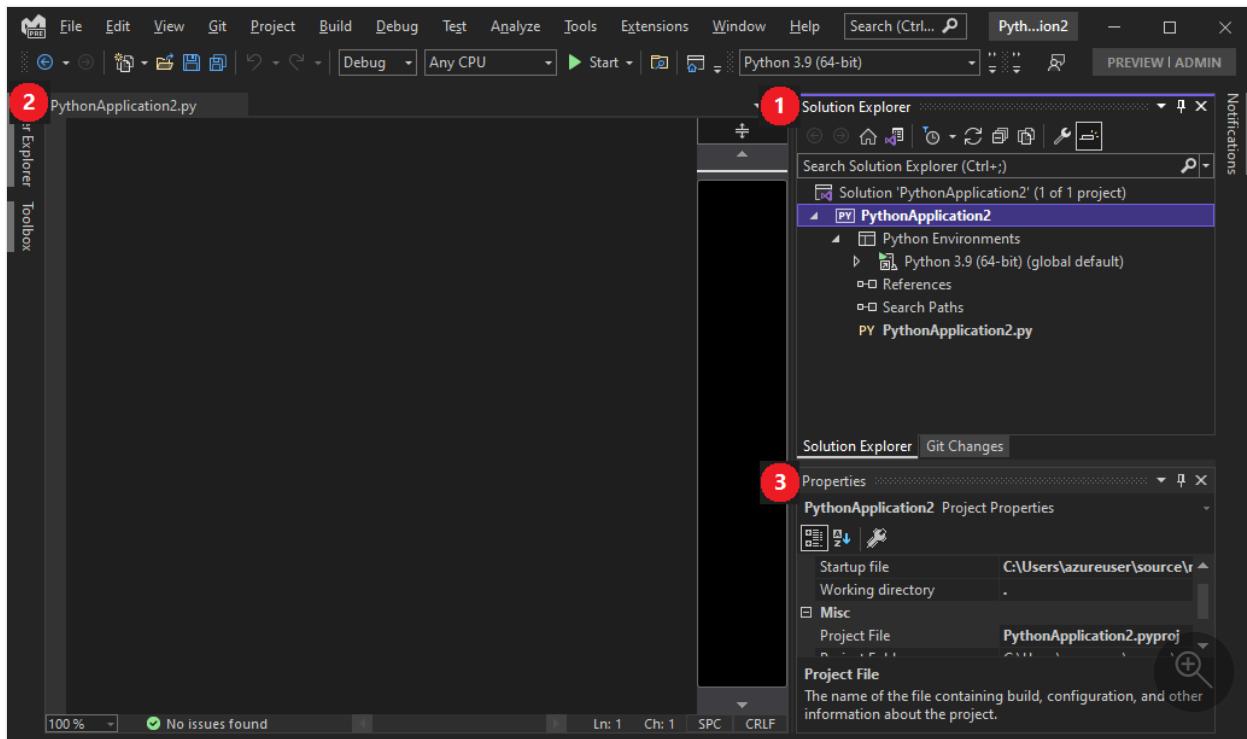
1. No Visual Studio, selecione **Arquivo > Novo > Projeto** ou use o atalho de teclado **Ctrl+Shift+N**. A tela **Criar um projeto** é exibida, e nela você pode pesquisar e procurar modelos em diferentes linguagens de programação.
2. Para exibir modelos do Python, pesquise *python*. O uso da pesquisa é uma ótima maneira de localizar um modelo quando você não se lembra da localização na árvore de linguagens.



O suporte à Web do Python no Visual Studio inclui vários modelos de projeto, como aplicativos Web nas estruturas Bottle, Flask e Django. Ao instalar o Python com o Instalador do Visual Studio, selecione **Supporte Web do Python** em **Opcional** para instalar esses modelos. Para este tutorial, comece com um projeto vazio.

3. Selecione o modelo **Aplicativo do Python** e depois **Avançar**.
4. Na tela **Configurar seu novo projeto**, especifique um nome e um local de arquivo para o projeto e selecione **Criar**.

Após alguns instantes, seu novo projeto será aberto no Visual Studio:

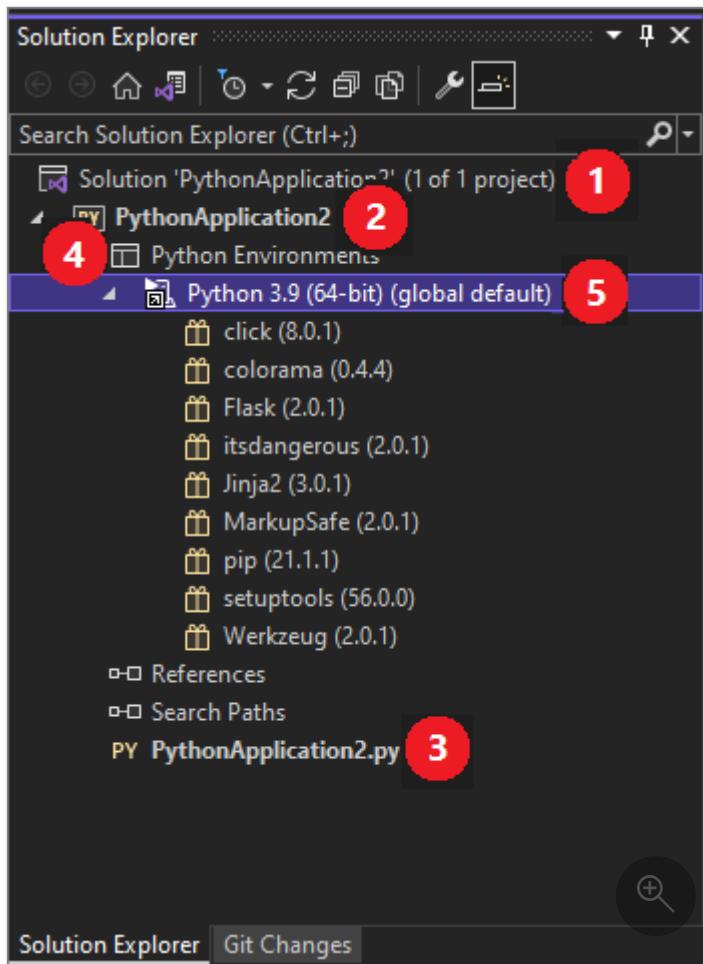


Isto é o que você vê:

- (1) A janela do **Gerenciador de Soluções** do Visual Studio mostra a estrutura do projeto.
- (2) O arquivo de código padrão será aberto no editor.
- (3) A janela **Propriedades** mostra mais informações para o item selecionado no **Gerenciador de Soluções**, incluindo a localização exata dele no disco.

Examinar elementos no Gerenciador de Soluções

Dedique algum tempo para se familiarizar com o **Gerenciador de Soluções**, que é o local em que você poderá procurar arquivos e pastas em seu projeto.



- (1) No nível superior está a **solução**, que, por padrão, tem o mesmo nome que seu projeto. Uma solução, representada por um arquivo `.sln` no disco, é um contêiner para um ou mais projetos relacionados. Por exemplo, se você escreve uma extensão de C++ para o seu aplicativo Python, o projeto de C++ poderá residir na mesma solução. A solução também poderá conter um projeto para um serviço Web, juntamente com projetos para programas de teste dedicados.
- (2) Seu projeto está realçado em negrito e usa o nome que você inseriu na caixa de diálogo **Criar um projeto**. No disco, esse projeto é representado por um arquivo `.pyproj` na pasta do projeto.
- (3) Em seu projeto, você vê arquivos de origem. Neste exemplo, você tem apenas um arquivo `.py`. Quando se seleciona um arquivo, as respectivas propriedades são exibidas na janela **Propriedades**. Se você não vir a janela **Propriedades**, selecione o ícone de chave inglesa no banner do **Gerenciador de Soluções**. Ao clicar duas vezes em um arquivo, ele será aberto da forma que for apropriada para esse arquivo.
- (4) No projeto também há o nó **Ambientes do Python**. Expanda o nó para mostrar os interpretadores do Python disponíveis.

- (5) Expanda um nó do interpretador para ver as bibliotecas que estão instaladas naquele ambiente.

Clique com o botão direito do mouse em qualquer nó ou item no **Gerenciador de Soluções** para mostrar um menu de contexto de comandos aplicáveis. Por exemplo, **Renomear** permite alterar o nome de um nó ou item, incluindo o projeto e a solução.

Próxima etapa

[Etapa 2: Escrever e executar o código](#)

Conteúdo relacionado

- [Projetos do Python no Visual Studio](#)
- [Saiba mais sobre a linguagem Python em python.org ↗](#)
- [Python para iniciantes ↗ \(python.org\)](#)

Etapa 2: Escrever e executar código

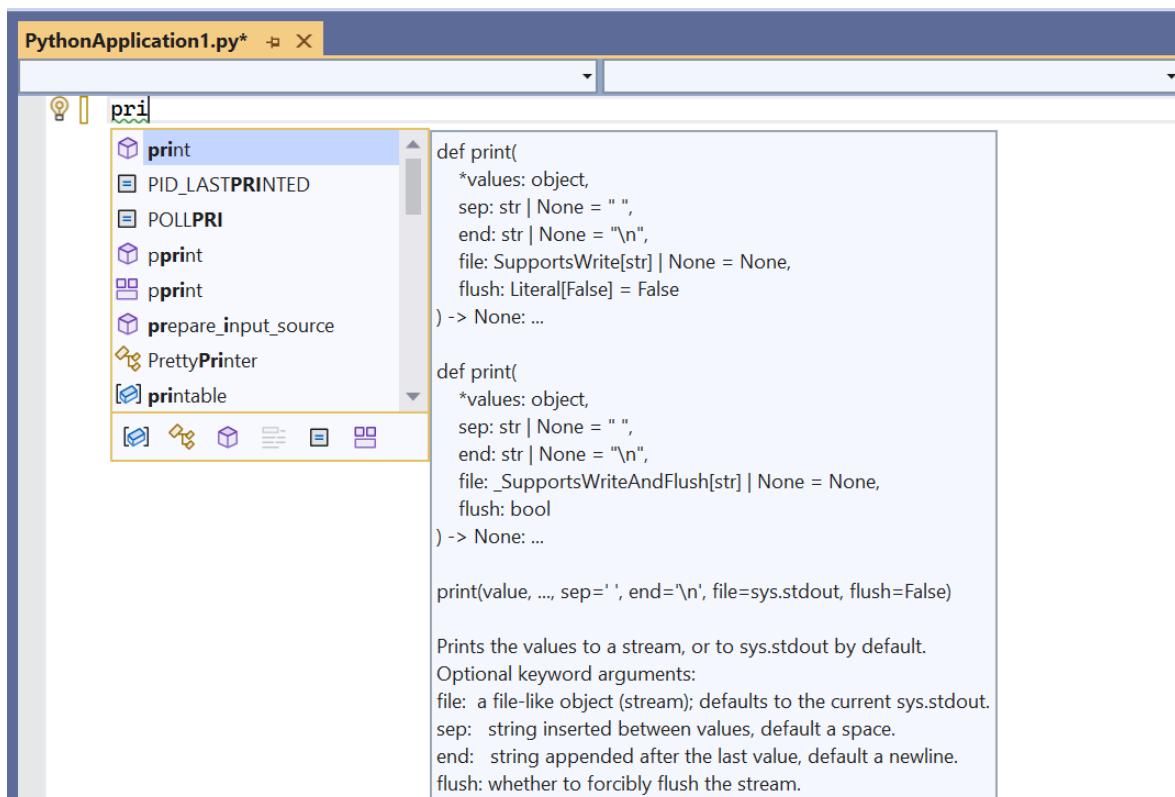
Artigo • 14/07/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

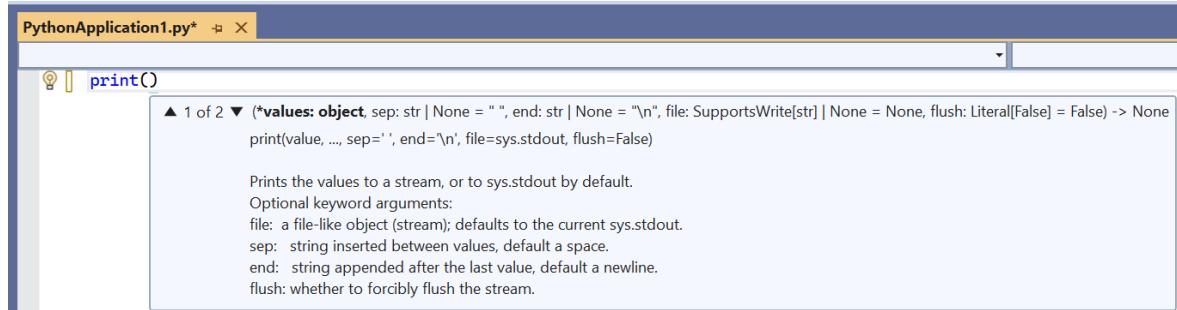
Etapa anterior: [Criar um novo projeto do Python](#)

Embora o **Gerenciador de Soluções** seja o local em que você gerencia arquivos de projeto, a janela do *editor* normalmente é o local em que você trabalha com o *conteúdo* dos arquivos, como o código-fonte. O editor está contextualmente ciente do tipo de arquivo que você está editando. O editor também reconhece a linguagem de programação (com base na extensão do arquivo) e oferece recursos apropriados para essa linguagem, como a coloração de sintaxe e o preenchimento automático usando o IntelliSense.

1. Quando você cria um novo projeto do "Aplicativo Python", um arquivo vazio padrão chamado *PythonApplication1.py* é aberto no editor do Visual Studio.
2. No editor, comece digitando `print("Hello, Visual Studio")` e observe como o Visual Studio IntelliSense exibe opções de preenchimento automático durante a digitação. A opção contornada na lista suspensa é o preenchimento padrão usado ao pressionar a tecla **Tab**. As conclusões são mais úteis quando instruções ou identificadores mais longos estão envolvidos.



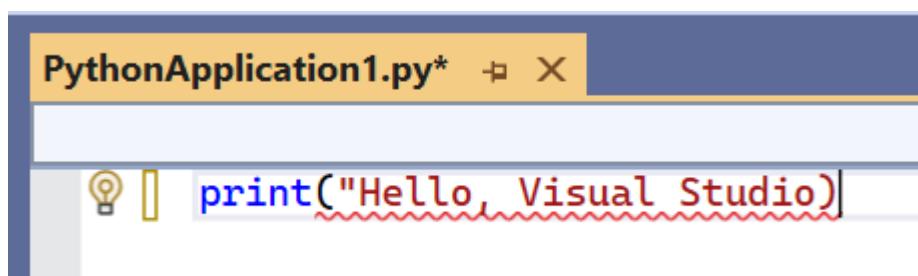
3. O IntelliSense mostra diferentes informações, com base da instrução que está sendo usada, da função que está sendo chamada e assim por diante. Com a função `print`, ao digitar `(` depois de `print` para indicar uma chamada função, as informações de uso completas dessa função são exibidas. O pop-up do IntelliSense também mostra o argumento atual em negrito (`valor` conforme mostrado aqui):



4. Preencha a instrução para que ela corresponda ao código abaixo:

```
Python
print("Hello, Visual Studio")
```

5. Observe a coloração de sintaxe que diferencia a instrução `print` do argumento `"Hello Visual Studio"`. Você pode excluir temporariamente a última `"` na cadeia de caracteres e observe como o Visual Studio mostra um sublinhado vermelho para o código que contém erros de sintaxe. Por fim, substitua o `"` para corrigir o código.

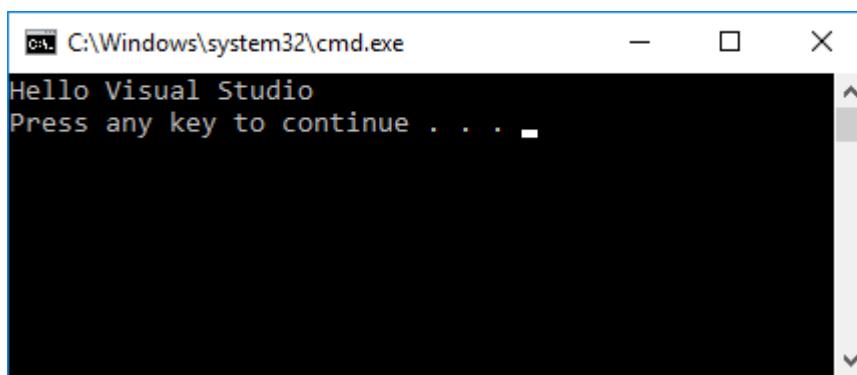


Dica

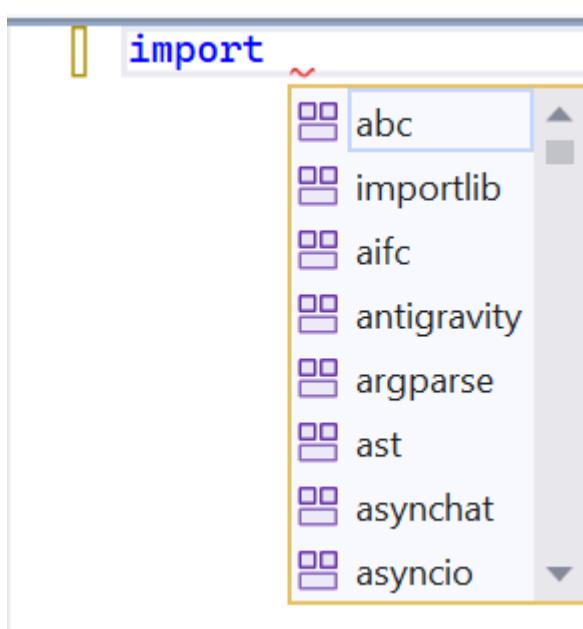
Como o ambiente de desenvolvimento é uma questão muito pessoal, o Visual Studio oferece controle total sobre a aparência e o comportamento do Visual Studio. Selecione o comando de menu **Ferramentas>Opções** e explore as configurações nas guias **Ambiente** e **Editor de Texto**. Por padrão, você vê somente um número limitado de opções; para ver todas as opções de todas

as linguagens de programação, selecione **Mostrar todas as configurações** na parte inferior da caixa de diálogo.

6. Execute o código que você escreveu até este ponto, pressionando **Ctrl+F5** ou selecionando o item de menu **Depurar>Iniciar Sem Depuração**. O Visual Studio avisará se ainda houver erros em seu código.
7. Quando você executar o programa, uma janela do console exibirá os resultados. É semelhante à execução de um interpretador do Python com *PythonApplication1.py* da linha de comando. Pressione qualquer tecla para fechar a janela e retornar ao editor do Visual Studio.

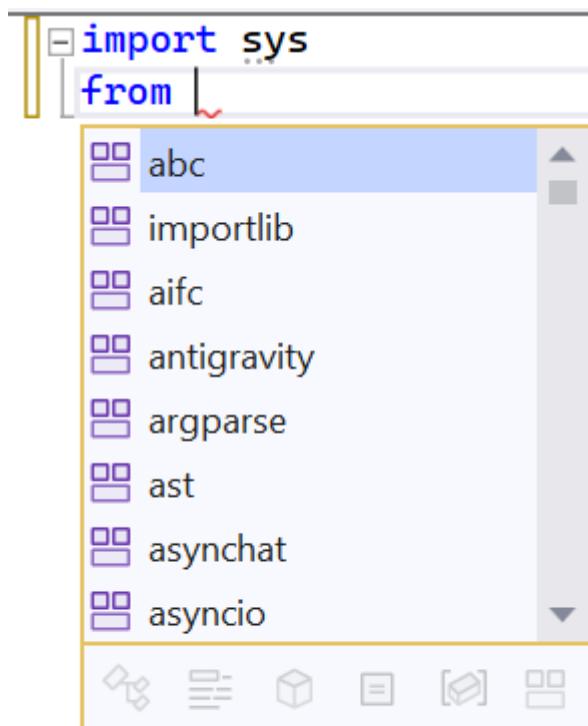


8. Além das conclusões para instruções e funções, o IntelliSense fornece preenchimentos para instruções `import` e `from` do Python. Esses preenchimentos ajudam você a descobrir com facilidade quais módulos estão disponíveis no ambiente e os membros desses módulos. No editor, exclua a linha `print` e comece a digitar `import`. Uma lista de módulos é exibida quando você digita o espaço:



9. Preencha a linha digitando ou selecionando `sys`.

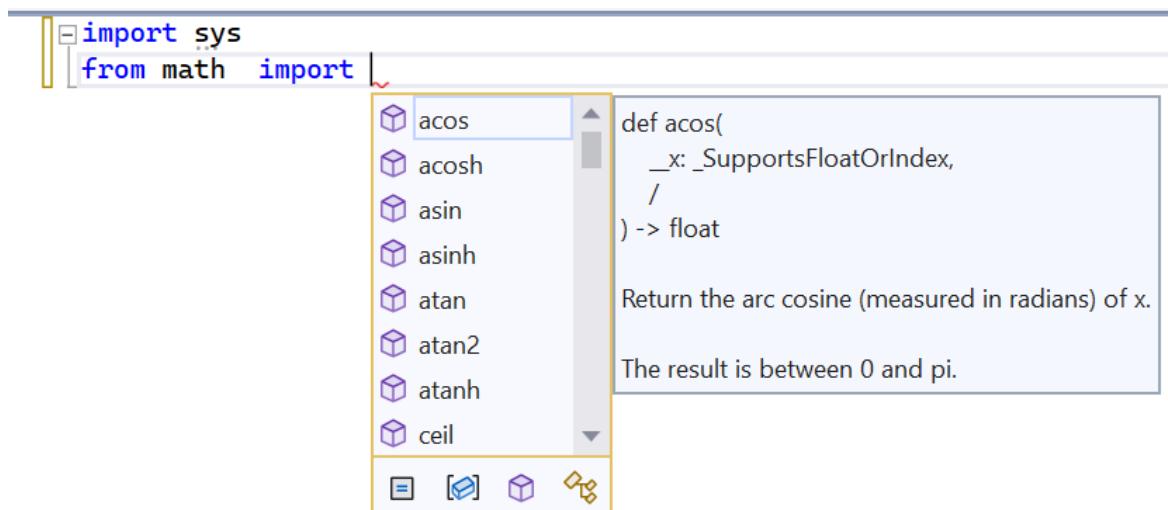
10. Na próxima linha, digite `from` para ver uma lista de módulos novamente:



```
import sys
from 
```

abc
importlib
aifc
antigravity
argparse
ast
asynchat
asyncio

11. Selecione ou digite `math` e continue digitando com um espaço e `import`, o que exibe os membros do módulo:



```
import sys
from math import 
```

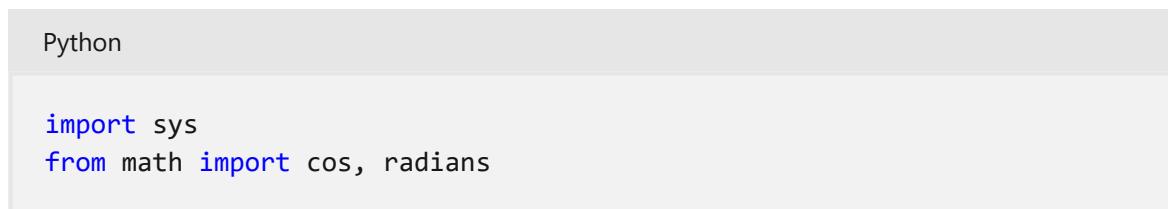
acos
acosh
asin
asinh
atan
atan2
atanh
ceil

def acos(
 _x: _SupportsFloatOrIndex,
 /
) -> float

Return the arc cosine (measured in radians) of x.

The result is between 0 and pi.

12. Conclua com a importação dos membros `cos` e `radians`, observando os preenchimentos automáticos disponíveis para cada um. Quando terminar, o código deverá ser exibido da seguinte maneira:



```
Python
```

```
import sys
from math import cos, radians
```

Dica

Os preenchimentos trabalham com subcadeias de caracteres durante a digitação, encontrando a correspondência de partes de palavras, letras no início de palavras e até mesmo caracteres ignorados. Confira [Editar o código – Preenchimentos](#) para obter detalhes.

13. Adicione um pouco mais de código para imprimir os valores de cosseno para 360 graus:

Python

```
for i in range(360):
    print(cos(radians(i)))
```

14. Execute o programa novamente com Ctrl+F5 ou **Depurar>Iniciar Sem Depuração**. Quando terminar, feche a janela de saída.

Próxima etapa

[Usar a janela interativa REPL](#)

Aprofunde-se um pouco mais

- [Editar código](#)
- [Código de formatação](#)
- [Refatorar o código](#)
- [Usar PyLint](#)

Etapa 3: usar a janela interativa REPL

Artigo • 20/07/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

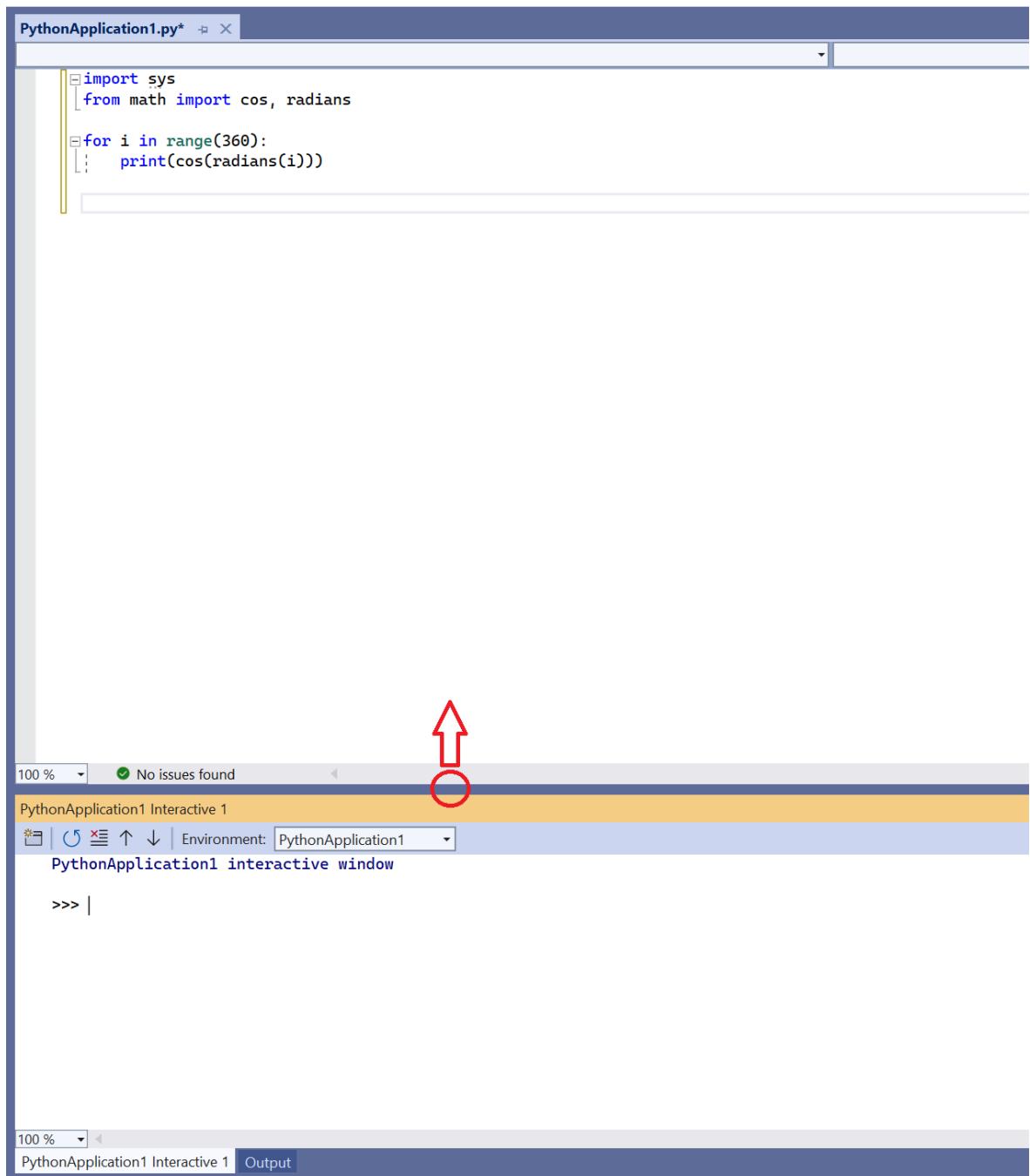
Etapa anterior: [Gravar e executar código](#)

A Janela Interativa do Visual Studio para Python oferece uma experiência avançada de REPL (leitura-avaliação-impressão-loop), que reduz consideravelmente o ciclo comum de edição-compilação-depuração. A Janela Interativa fornece todos os recursos da experiência de REPL da linha de comando do Python. Ela também facilita a troca de código com arquivos de origem no editor do Visual Studio, o que seria complicado com a linha de comando.

Observação

Para problemas com REPL, verifique se os pacotes `ipython` e `ipykernel` estão instalados e, para obter ajuda na instalação dos pacotes, confira a [guia de pacotes de ambientes Python](#).

1. Abra a Janela Interativa clicando com o botão direito do mouse no ambiente de projeto do Python no Gerenciador de Soluções (como **Python 3.6 (32 bits)**, mostrado em um gráfico anterior) e selecionando **Abrir Janela Interativa**. Como alternativa, você pode selecionar **Exibir>Outras Janelas>Janelas Interativas do Python** no menu principal do Visual Studio.
2. A Janela Interativa abre-se abaixo do editor com o prompt padrão de REPL do Python `>>>`. A lista suspensa **Ambiente** permite selecionar um intérprete específico com o qual trabalhar. Se você quiser aumentar a Janela Interativa, poderá arrastar o separador entre as duas janelas, como mostrado na imagem abaixo:



Dica

Você pode redimensionar todas as janelas no Visual Studio, arrastando os separadores de bordas. Você também pode arrastar e retirar janelas para fora do quadro do Visual Studio e reorganizá-las da forma que quiser dentro do quadro. Para obter detalhes completos, confira [Personalizar layouts de janela](#).

3. Insira algumas instruções, como `print("Hello, Visual Studio")`, e expressões, como `123/456`, para ver resultados imediatos:

The screenshot shows the Python Application window with the title "PythonApplication1 Interactive 1". The top bar includes icons for file operations and tabs for "Environment: PythonApplication1" and "Module: _main_". Below the bar, the text "PythonApplication1 interactive window" is displayed. A code snippet is shown in the window:

```
>>> print("Hello, Visual Studio")
Hello, Visual Studio
>>> 123/456
0.26973684210526316
>>>
```

The bottom of the window has a zoom level of "100%" and tabs for "Output" and "PythonApplication1 Interactive 1".

4. Ao começar a escrever uma instrução de várias linhas (como uma definição de função) a **Janela Interativa** mostrará o prompt ... do Python para continuar as linhas. Ao contrário do REPL de linha de comando, isso proporcionará recuo automático. Você pode adicionar uma nova linha ... pressionando Shift+Enter:

The screenshot shows the Python Application window with the title "PythonApplication1 Interactive 1". The top bar includes icons for file operations and tabs for "Environment: PythonApplication1" and "Module: _main_". Below the bar, the text "PythonApplication1 interactive window" is displayed. A code snippet is shown in the window:

```
>>> print("Hello, Visual Studio")
Hello, Visual Studio
>>> 123/456
0.26973684210526316
>>> def f(num):
...     print(num*2)
...
>>> f(17)
34
>>> |
```

The bottom of the window has a zoom level of "100%" and tabs for "Output" and "PythonApplication1 Interactive 1".

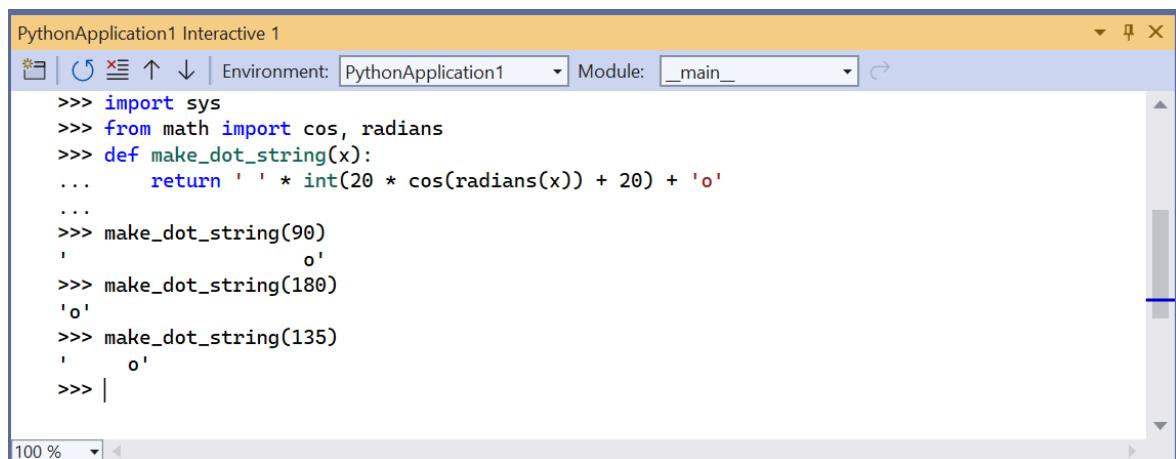
5. A **Janela Interativa** fornece um histórico completo de tudo o que foi inserido, e é uma melhoria do REPL de linha de comando com itens de histórico de várias linhas. Por exemplo, com facilidade, é possível cancelar toda a definição da função `f` como uma única unidade e alterar o nome para `make_double`, em vez de recriar a função linha por linha.
6. O Visual Studio pode enviar várias linhas de código de uma janela do editor para a **Janela Interativa**. Essa capacidade permite que você mantenha o código em um arquivo de origem e envie facilmente fragmentos selecionados para a **Janela Interativa**. Assim, você poderá trabalhar com esses fragmentos de código no ambiente REPL rápido em vez de ter que executar o programa inteiro. Para ver esse recurso, primeiro substitua o loop `for` no arquivo `PythonApplication1.py` pelo código abaixo:

Python

```
# Create a string with spaces proportional to a cosine of x in degrees
def make_dot_string(x):
    return ' ' * int(20 * cos(radians(x)) + 20) + 'o'
```

7. Selecione as instruções de função `import`, `from` e `make_dot_string` no arquivo `.py`.

Clique com o botão direito do mouse no texto selecionado e escolha **Enviar para Interativo** (ou pressione **Ctrl+Enter**). O fragmento de código será imediatamente colado na **Janela Interativa** e executado. Como o código definiu uma função, é possível testar essa função rapidamente chamando-a algumas vezes:



The screenshot shows the Python Application window with the title "PythonApplication1 Interactive 1". The interface includes a toolbar with icons for file operations, a status bar, and dropdown menus for Environment and Module. The main area contains an interactive Python shell. The user has typed the following code:

```
>>> import sys
>>> from math import cos, radians
>>> def make_dot_string(x):
...     return ' ' * int(20 * cos(radians(x)) + 20) + 'o'
...
>>> make_dot_string(90)
' '
'o'
>>> make_dot_string(180)
'o'
>>> make_dot_string(135)
' '
'o'
>>> |
```

Dica

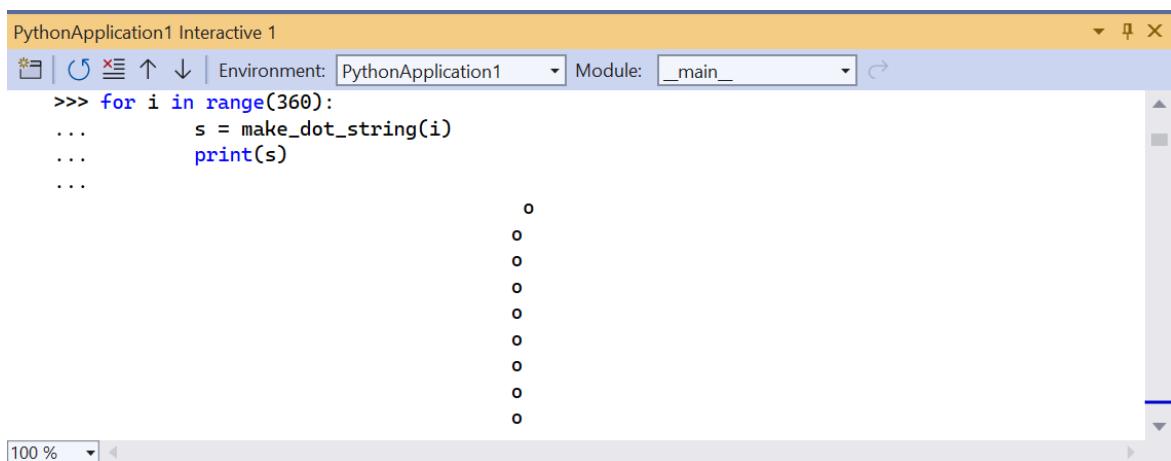
O uso de **Ctrl+Enter** no editor *sem* uma seleção, executará a linha de código atual na **Janela Interativa** e posicionará automaticamente o cursor na próxima linha. Com esse recurso, ao pressionar **Ctrl+Enter** repetidamente, você terá uma maneira conveniente de percorrer o código, o que não é possível somente com a linha de comando do Python. Isso também permitirá que você percorra o código sem executar o depurador e sem, necessariamente, começar desde o início do programa.

8. Você também pode copiar e colar várias linhas de código de qualquer fonte na **Janela Interativa**, como no snippet a seguir, o que é difícil fazer com o REPL da linha de comando do Python. Ao colar, a **Janela Interativa** executa o código como se você o tivesse digitado:

Python

```
for i in range(360):
    s = make_dot_string(i)
```

```
print(s)
```



The screenshot shows the Python Application 1 Interactive window. The code in the editor is:

```
>>> for i in range(360):
...     s = make_dot_string(i)
...     print(s)
...

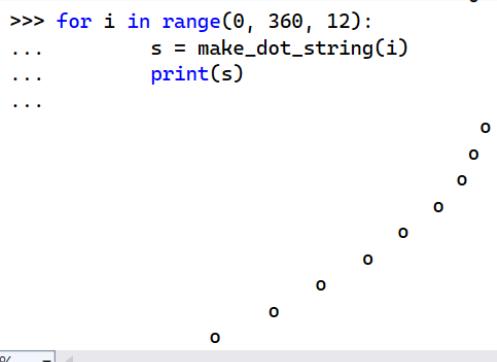
```

The output window displays a vertical column of 360 small circles ('o') representing the curve of the cosine function over 360 degrees.

9. Como você pode ver, esse código funciona bem, mas a saída não é muito impressionante. Um valor de etapa diferente no loop `for` mostraria mais da curva do cosseno. Todo o loop `for` está disponível no histórico de REPL como uma única unidade. Você pode voltar e fazer as alterações desejadas e depois testar a função novamente. Pressione a seta para cima para, primeiro, recuperar o loop `for`. Você pode navegar no código pressionando as setas para a esquerda e para a direita (até que você faça isso, as setas para baixo e para cima continuam a percorrer o histórico). Navegue até a especificação `range` e altere-a para `range(0, 360, 12)`. Em seguida, pressione **Ctrl+Enter** em qualquer lugar no código para executar toda a instrução novamente:

```
>>> for i in range(0, 360, 12):
...     s = make_dot_string(i)
...     print(s)
...

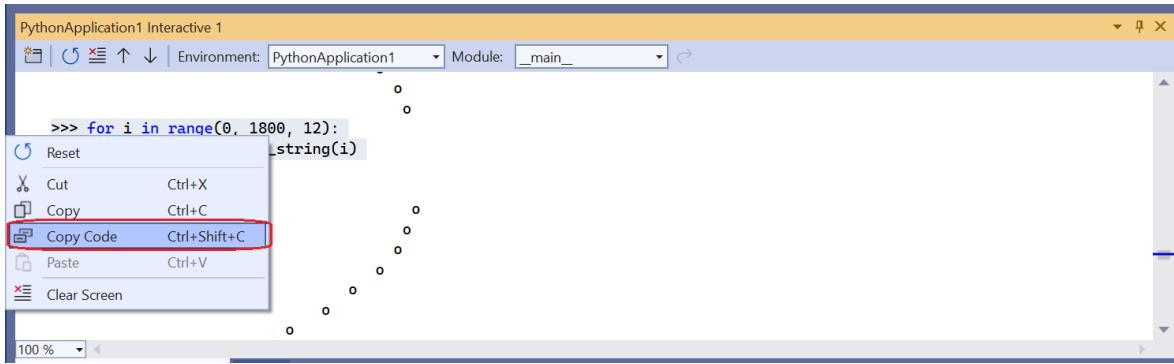
```



The screenshot shows the Python Application 1 Interactive window. The code is identical to the previous one, but the output shows a much denser and more continuous curve of dots, indicating a larger number of points plotted due to the smaller step size of 12 degrees.

10. Repita o processo para fazer experiências com configurações de etapas diferentes, até encontrar um valor que você mais goste. Você também pode fazer a curva se repetir, aumentando o intervalo, por exemplo, `range(0, 1800, 12)`.
11. Quando estiver satisfeito com o código que você escreveu na Janela Interativa, selecione-o. Em seguida, clique com o botão direito do mouse no código e escolha **Copiar Código (Ctrl+Shift+C)**. Por fim, cole o código selecionado no editor. Observe como esse recurso especial do Visual Studio omite automaticamente qualquer saída, bem como os prompts `>>>` e `....`. Por exemplo,

A imagem abaixo mostra o uso do comando **Copiar Código** em uma seleção que inclui os prompts e a saída:



Ao colar no editor, você obtém somente o código:

A screenshot of a code editor window titled "Python". The code editor contains the following Python code:

```
for i in range(0, 1800, 12):
    s = make_dot_string(i)
    print(s)
```

Se quiser copiar o conteúdo exato da **Janela Interativa**, incluindo os prompts e a saída, basta usar o comando padrão **Copiar**.

12. O que você acabou de fazer é usar o ambiente de REPL rápido da **Janela Interativa** para planejar os detalhes de uma pequena parte de código e depois adicionou convenientemente esse código ao arquivo de origem do seu projeto. Agora, ao executar o código novamente com **Ctrl+F5** (ou **Depurar>Iniciar sem Depurar**), você verá exatamente os resultados desejados.

Próxima etapa

[Executar o código no depurador](#)

Aprofunde-se um pouco mais

- [Usar a Janela Interativa](#)
- [Usar o IPython REPL](#)

Etapa 4: Executar o código no depurador

Artigo • 11/08/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [usar a janela interativa REPL](#)

O Visual Studio fornece recursos para gerenciar projetos, uma experiência de edição avançada, a janela **Interativa** e a depuração completa para código Python. No depurador, você pode executar seu código passo a passo, incluindo cada iteração de um loop. Você também pode pausar o programa sempre que determinadas condições são verdadeiras. A qualquer momento em que o programa estiver em pausa no depurador, você poderá examinar todo o estado do programa e alterar o valor de variáveis. Essas ações são essenciais para a localização de bugs do programa e também fornecem recursos úteis para seguir o fluxo exato do programa.

1. Substitua o código no arquivo *PythonApplication1.py* pelo código a seguir. Essa variação do código expande o `make_dot_string` para que você possa examinar as etapas distintas no depurador. Ela também coloca o loop `for` em uma função `main` e executa-o explicitamente, chamando essa função:

Python

```
from math import cos, radians

# Create a string with spaces proportional to a cosine of x in degrees
def make_dot_string(x):
    rad = radians(x)                                # cos works with
    radians
    numspaces = int(20 * cos(rad) + 20)            # scale to 0-40 spaces
    st = ' ' * numspaces + 'o'                      # place 'o' after the
    spaces
    return st

def main():
    for i in range(0, 1800, 12):
        s = make_dot_string(i)
        print(s)

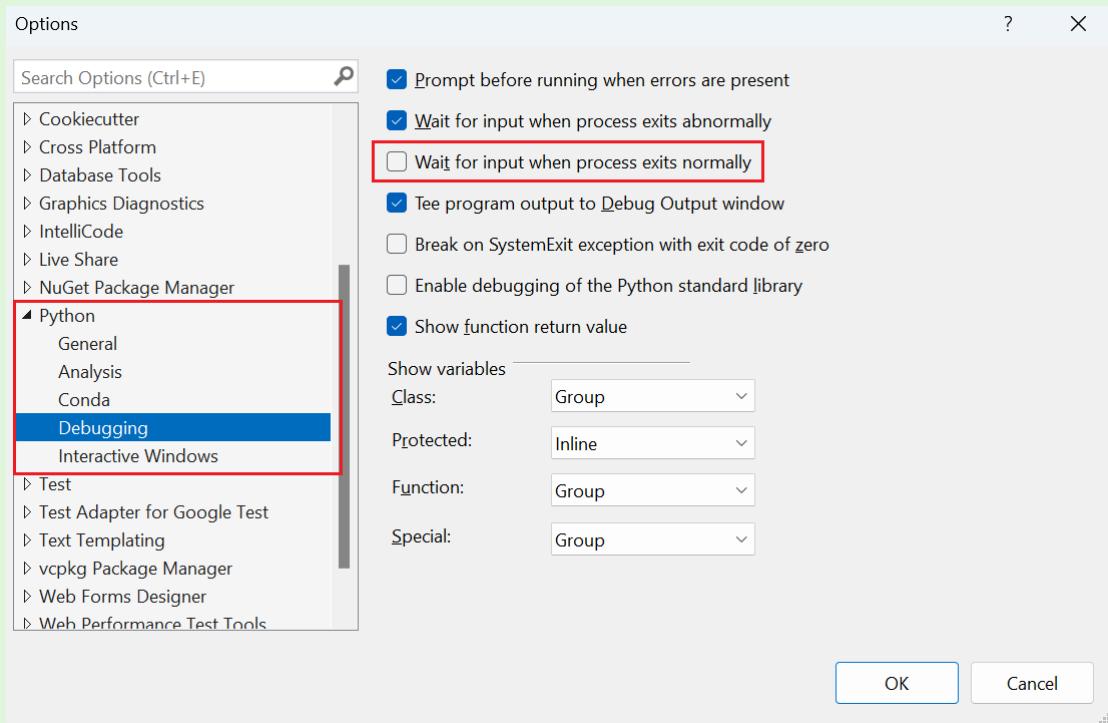
main()
```

2. Verifique se o código funciona corretamente pressionando **F5** ou selecionando o comando de menu **Depurar>Iniciar Depuração**. Esse comando executa o código

no depurador. Até agora, nada foi feito para pausar o programa enquanto ele está em execução; ele apenas imprimirá um padrão de onda para algumas iterações. Pressione qualquer tecla para fechar a janela de saída.

Dica

Para fechar a Janela de Saída automaticamente quando o programa for concluído, selecione o comando de menu **Ferramentas>Opções**, expanda o nó do **Python**, selecione **Depuração** e, em seguida, desmarque a opção **Aguardar pela entrada quando o processo for encerrado normalmente**:



Para obter mais informações sobre depuração e como definir argumentos de script e interpretador, confira [Depurar seu código Python](#).

3. Defina um ponto de interrupção na instrução `for` clicando uma vez na margem cinza próxima a essa linha ou colocando o cursor na linha e usando o comando **Depurar>Ativar/Desativar Ponto de Interrupção (F9)**. Um ponto vermelho é exibido na margem cinza para indicar o ponto de interrupção (conforme indicado pela seta abaixo):

```

def main():
    for i in range(0, 1800, 12):
        s = make_dot_string(i)
        print(s)

main()

```

4. Inicie o depurador novamente (**F5**) e veja que a execução do código é interrompida na linha com o ponto de interrupção. Aqui você pode inspecionar a pilha de chamadas e examinar variáveis. As variáveis que estão no escopo aparecem na janela **Autos** quando elas estão definidas. Você também pode alternar para a exibição **Locais** na parte inferior dessa janela para mostrar todas as variáveis que o Visual Studio localiza no escopo atual (incluindo funções), antes mesmo que elas sejam definidas:

```

from math import cos, radians

# Create a string with spaces proportional to a cosine of x in degrees
def make_dot_string(x):
    rad = radians(x)                                # cos works with radians
    numspaces = int(20 * cos(rad) + 20)             # scale to 0-40 spaces
    st = ' ' * numspaces + 'o'                      # place 'o' after the spaces
    return st

def main():
    for i in range(0, 1800, 12):
        s = make_dot_string(i)
        print(s)

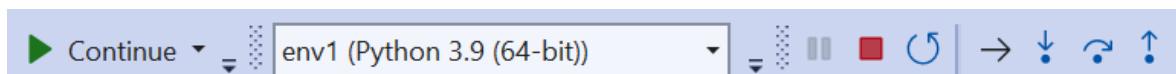
main()

```

| Locals | | |
|-------------|-------|------|
| Name | Value | Type |
| { } Globals | | |

| Call Stack | | |
|------------------|--------|--|
| Name | Lang | |
| _main__main : 11 | Pyt... | |
| _main_ : 15 | Pyt... | |

5. Observe a barra de ferramentas de depuração (mostrada abaixo) na parte superior da janela do Visual Studio. Essa barra de ferramentas fornece acesso rápido aos comandos de depuração mais comuns (que também podem ser encontrados no menu **Depurar**):



Os botões, da esquerda para a direita, são os seguintes:

| Botão | Comando |
|--------------------------------------|--|
| Continuar (F5) | Executa o programa, até o próximo ponto de interrupção ou até a conclusão do programa. |
| Interromper Tudo (Ctrl+Alt+Break) | Pausa um programa de longa execução. |
| Parar Depuração (Shift+F5) | Interrompe o programa onde quer que esteja e sai do depurador. |
| Reiniciar (Ctrl+Shift+F5) | Interrompe o programa no ponto em que está e o reinicia no depurador. |
| Mostrar Próxima Instrução (Alt+Num*) | Altera para a próxima linha de código a ser executada. Isso é útil quando você navega em seu código durante uma sessão de depuração e deseja retornar rapidamente ao ponto em que o depurador está em pausa. |
| Entrar em (F11) | Executa a próxima linha de código, entrando em funções chamadas. |
| Passo a passo (F10) | Executa a próxima linha de código sem entrar em funções chamadas. |
| Sair (Shift+F11) | Executa o restante da função atual e pausa no código de chamada. |

6. Contorne a instrução `for` usando **Contornar**. *Passo a passo* significa que o depurador executa a linha de código atual, incluindo todas as chamadas de função e, em seguida, imediatamente entra em pausa outra vez. Observe, no código, como a variável `i` agora está definida nas janelas **Locais** e **Autos**.
7. Contorne a próxima linha de código, que chama `make_dot_string` e entra em pausa. Aqui o **Contorno** significa especificamente que o depurador executa todo o `make_dot_string` e entra em pausa ao retornar. O depurador não é interrompido dentro dessa função, a menos que exista nela outro ponto de interrupção.
8. Continue depurando o código passo a passo mais algumas vezes e observe como os valores na janela **Locais** ou **Autos** se alteram.
9. Na janela **Locais** ou **Autos**, clique duas vezes na coluna **Valor** das variáveis `i` ou `s` para editar o valor. Pressione **Enter** ou selecione uma área fora desse valor para aplicar alterações, se houver.
10. Continuar percorrendo o código passo a passo, usando **Intervir**. **Intervir** significa que o depurador entra dentro de qualquer chamada de função para a qual ele

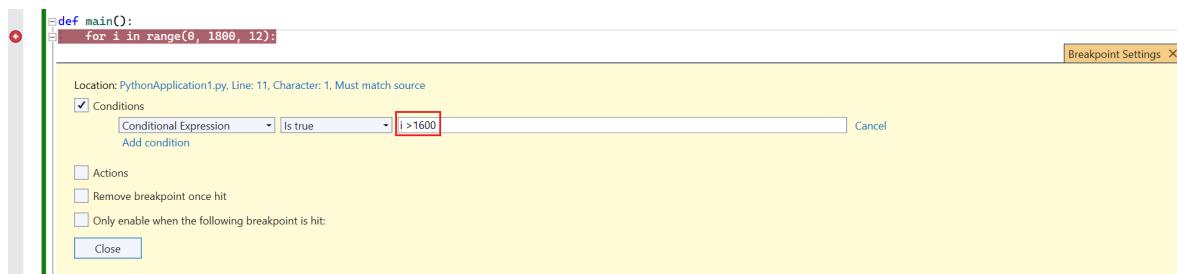
tenha informações de depuração, como `make_dot_string`. Uma vez dentro do `make_dot_string`, você pode examinar as variáveis locais e percorrer o código especificamente.

11. Continue depurando passo a passo com **Intervir** e observe que, ao chegar ao fim do `make_dot_string`, a próxima etapa retorna para o loop `for` com o novo valor retornado na variável `s`. Conforme você avança novamente para a instrução `print`, observe que o **Intervir** em `print` não entra nessa função. Isso ocorre porque `print` não está escrita em Python, mas é código nativo dentro do runtime do Python.

12. Continue usando **Intervir** até que você esteja novamente quase em `make_dot_string`. Então, use **Sair** e observe que você retornará ao loop `for`. Com **Sair**, o depurador executa o restante da função e faz automaticamente uma pausa no código de chamada. Isso é útil quando você percorre alguma parte de uma função longa que deseja depurar. Ele percorrerá o restante e não definirá um ponto de interrupção explícito no código de chamada.

13. Para continuar a execução do programa até que o próximo ponto de interrupção seja atingido, use **Continuar (F5)**. Como há um ponto de interrupção no loop `for`, você interrompe na próxima iteração.

14. Percorrer centenas de iterações de um loop pode ser entediante, portanto, o Visual Studio permite que você adicione uma *condição* a um ponto de interrupção. Assim, o depurador só pausa o programa no ponto de interrupção quando a condição é satisfeita. Por exemplo, você pode usar uma condição com o ponto de interrupção na instrução `for` para que ele faça uma pausa somente quando o valor de `i` exceder 1600. Para definir a condição, clique com o botão direito do mouse no ponto de interrupção e selecione **Condições (Alt+F9>C)**. Na janela pop-up **Configurações de Ponto de Interrupção** exibida, insira `i > 1600` como a expressão e selecione **Fstrar**. Pressione **F5** para continuar e observe que o programa executa muitas iterações antes da próxima interrupção.



15. Para executar o programa até a conclusão, desabilite o ponto de interrupção clicando com o botão direito do mouse no ponto na margem e selecionando

Desabilitar ponto de interrupção (Ctrl+F9). Em seguida, selecione **Continuar** (ou pressione **F5**) para executar o programa. Quando o programa for finalizado, o Visual Studio interromperá a sessão de depuração e retornará para o modo de edição. Você também pode excluir o ponto de interrupção selecionando seu ponto ou clicando com o botão direito do mouse no ponto e selecionando **Excluir ponto de interrupção**. Ele também exclui qualquer condição que você tenha definido anteriormente.

💡 Dica

Em algumas situações, como uma falha ao iniciar o interpretador do Python em si, a janela de saída poderá aparecer apenas rapidamente e fechar-se automaticamente, sem dar uma oportunidade de ver as mensagens de erros. Se isso acontecer, clique com botão direito do mouse no projeto no **Gerenciador de Soluções**, selecione **Propriedades**, selecione a guia **Depurar** e adicione `-i` ao campo **Argumentos do Interpretador**. Esse argumento faz com que o interpretador entre no modo interativo após a conclusão de um programa, mantendo a janela aberta até que você pressione **Ctrl+Z>Enter** para sair.

Próxima etapa

[Instalar pacotes no ambiente do Python](#)

Aprofunde-se um pouco mais

- [Depuração](#)
- [Depuração no Visual Studio](#) oferece uma documentação completa sobre os recursos de depuração do Visual Studio.

Etapa 5: Instalar pacotes no ambiente do Python

Artigo • 14/03/2023

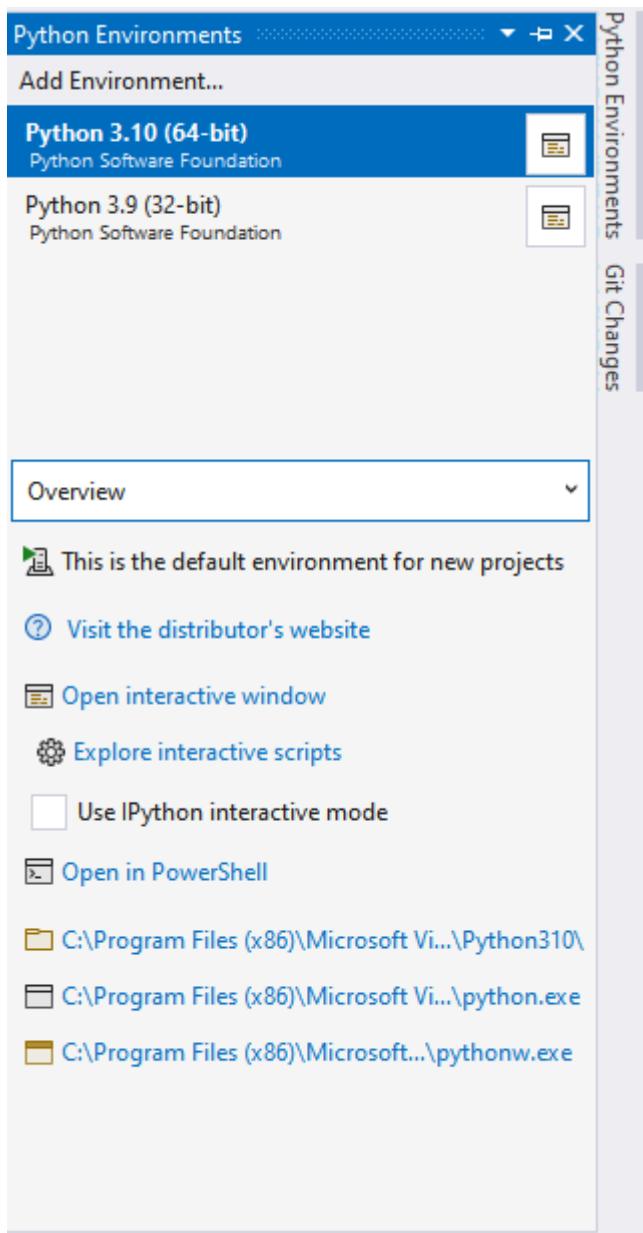
Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [executar o código no depurador](#)

A comunidade de desenvolvedores do Python produziu milhares de pacotes úteis que você pode incorporar em seus próprios projetos. O Visual Studio oferece uma interface do usuário para gerenciar pacotes em seus ambientes do Python.

Exibir ambientes

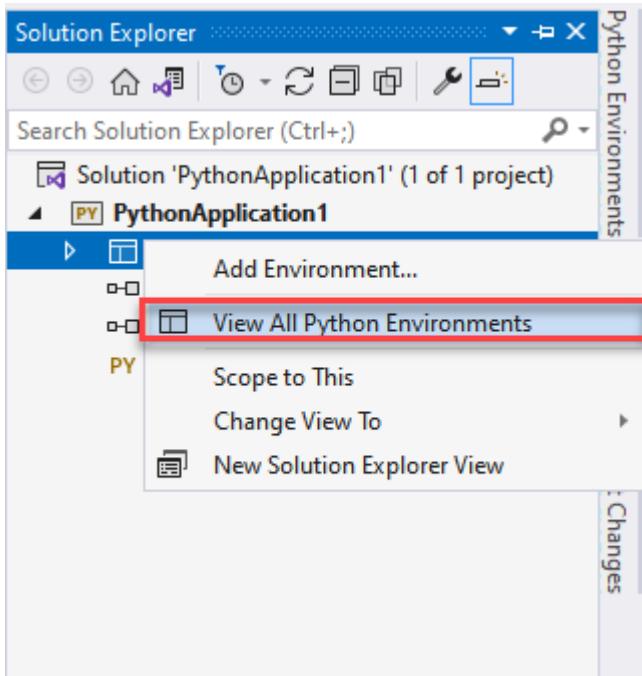
1. Selecione o comando de menu **Exibir>Outras Janelas>Ambientes do Python**. A janela **Ambientes de Python** será aberta como um par com o **Gerenciador de Soluções** e mostrará os diferentes ambientes disponíveis para você. A lista mostra ambos os ambientes em que você instalou usando o instalador do Visual Studio e aqueles que você instalou separadamente. Isso inclui ambientes globais, virtuais e do conda. O ambiente em negrito é o ambiente padrão, que é usado para novos projetos. Para obter mais informações sobre como trabalhar com ambientes, confira [Como criar e gerenciar ambientes do Python em ambientes do Visual Studio](#).



Observação

Você também pode usar o atalho de teclado **Ctrl+K, Ctrl+`** para abrir a janela **Ambientes do Python** da janela do Gerenciador de Soluções. Se o atalho não funcionar e você não encontrar a janela Ambientes do Python no menu, é possível que você não tenha instalado a carga de trabalho do Python. Confira [Como instalar o suporte do Python no Visual Studio no Windows](#) para obter diretrizes sobre como instalar o Python.

Com um projeto do Python aberto, você pode abrir a janela **Ambientes do Python** por meio do **Gerenciador de Soluções**. Clique com o botão direito do mouse em **Ambientes do Python** e selecione **Exibir Todos os Ambientes do Python**.



2. Agora, crie um projeto com **Arquivo>Novo >Projeto**, escolhendo o modelo **Aplicativo do Python**.
3. No arquivo de código que é exibido, cole o código a seguir, que cria uma curva de coseno como nas etapas anteriores do tutorial, só que desta vez, plotada graficamente. Você também pode usar o projeto criado anteriormente e substituir o código.

Python

```
from math import radians
import numpy as np # installed with matplotlib
import matplotlib.pyplot as plt

def main():
    x = np.arange(0, radians(1800), radians(12))
    plt.plot(x, np.cos(x), 'b')
    plt.show()

main()
```

4. Na janela do editor, passe o mouse sobre as instruções de importação `numpy` e `matplotlib`. Você observará que eles não foram resolvidos. Para resolver as instruções de importação, instale os pacotes no ambiente global padrão.

```
from math import radians
import numpy as np      # installed with matplotlib
import matplotlib.pyplot as plt

def main():
    x = np
    plt.p
    plt.show()
```

A tooltip is displayed over the `plt.show()` line, stating: "reportMissingModuleSource: Import "matplotlib.pyplot" could not be resolved from source".

5. Quando olhar para a janela do editor, observe que quando você passa o mouse sobre as instruções de importação `numpy` e `matplotlib`, elas não são resolvidas. O motivo é que os pacotes não foram instalados no ambiente global padrão.

Por exemplo, escolha **Abrir janela Interativa** e uma janela **Interativa** desse ambiente específico será exibida no Visual Studio.

6. Use a lista suspensa abaixo da lista de ambientes para alternar para a guia **Pacotes**. A guia **Pacotes** lista todos os pacotes que estão atualmente instalados no ambiente.

Instalar pacotes usando a janela Ambientes do Python

1. Na janela Ambientes do Python, selecione o ambiente padrão para novos projetos do Python e escolha a guia **Pacotes**. Em seguida, você verá uma lista de pacotes que estão atualmente instalados no ambiente.

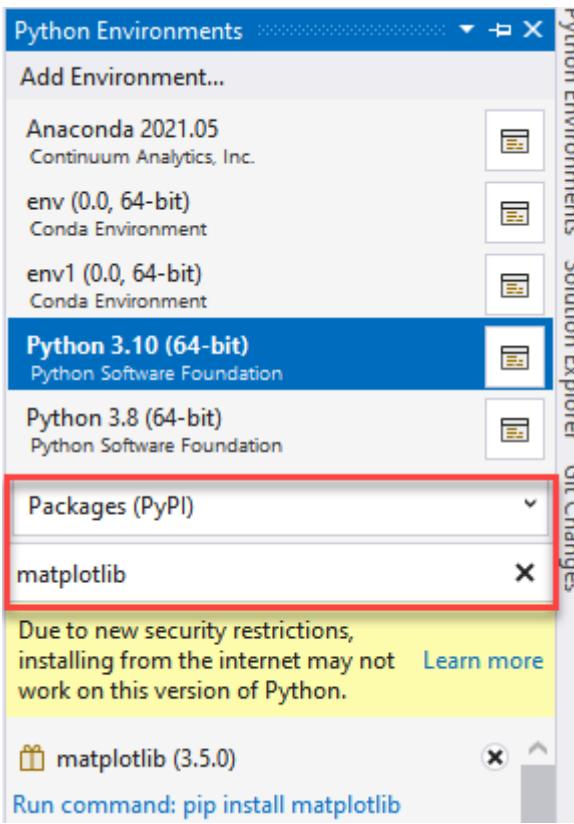
[Instalar pacotes em um ambiente](#)

2. Instale `matplotlib` inserindo o nome dele no campo de pesquisa e, em seguida, selecionando a opção **Executar comando**: `pip install matplotlib`. A execução do comando instalará `matplotlib` e todos os pacotes dos quais ele depende (nesse caso, incluindo `numpy`).

3. Escolha a guia **Pacotes**.

4. Dê permissão para a elevação, se for solicitado.

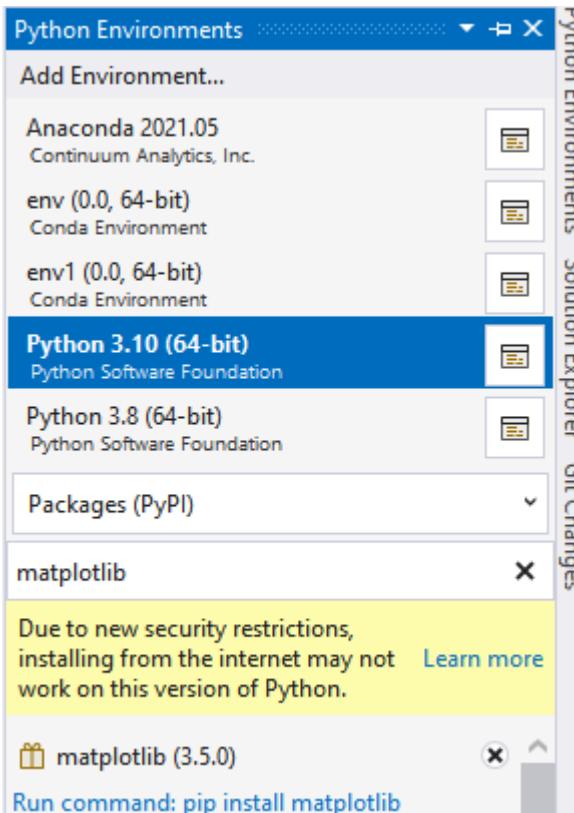
5. Depois de instalar o pacote, ele aparecerá na janela **Ambientes de Python**. O X à direita do pacote serve para desinstalá-lo.



6. Dê permissão para a elevação, se for solicitado.

7. Depois que o pacote for instalado, ele aparecerá na janela Ambientes de Python.

O X à direita do pacote serve para desinstalá-lo.



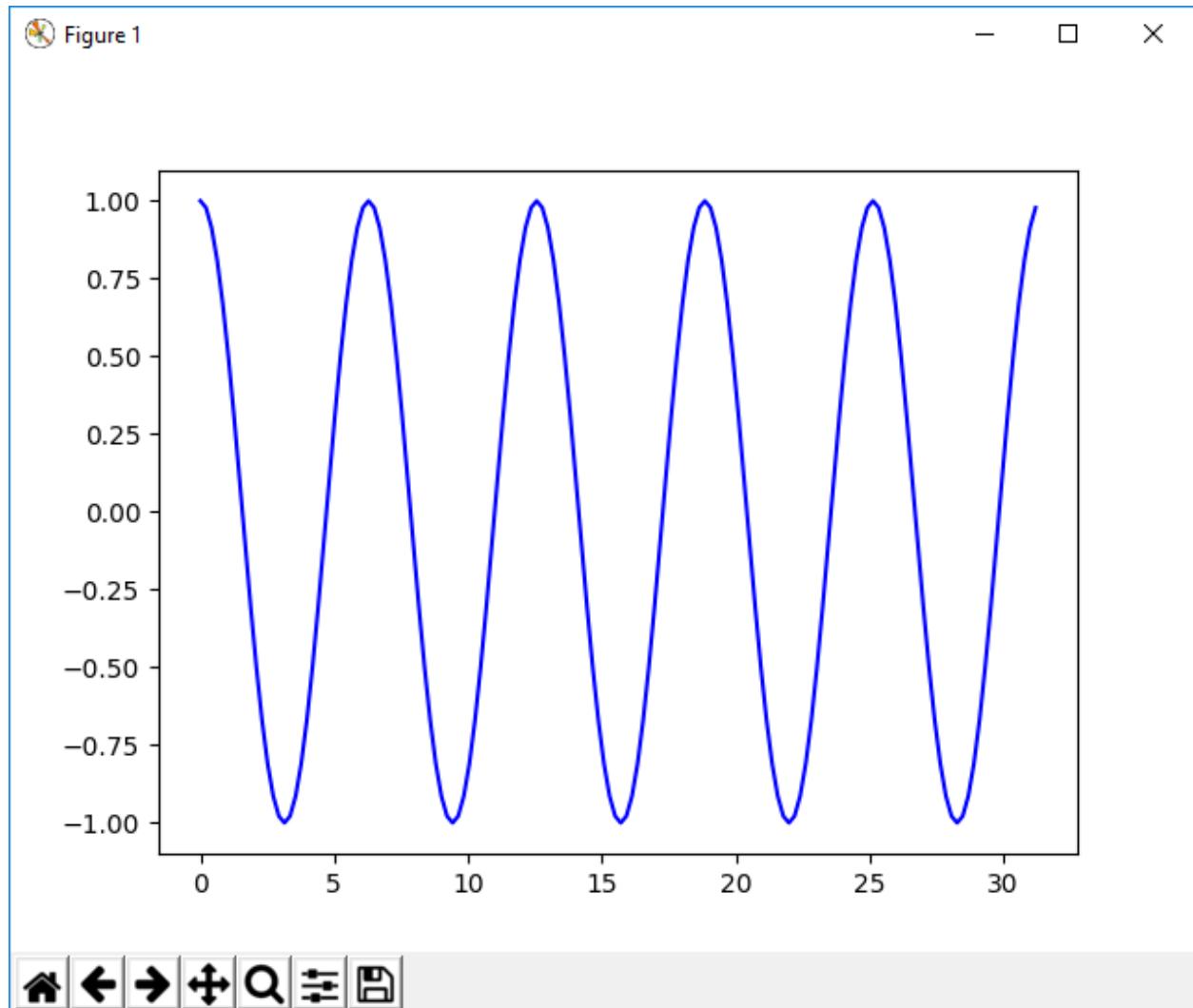
! Observação

Uma pequena barra de progresso pode ser exibida sob o ambiente para indicar que o Visual Studio está criando o banco de dados do IntelliSense para o pacote recém-instalado. A guia **IntelliSense** também mostra mais informações detalhadas. Observe que, até que o banco de dados seja concluído, os recursos do IntelliSense, como preenchimento automático e verificação de sintaxe, não estarão ativos no editor para esse pacote.

O Visual Studio 2017 versão 15.6 e posterior usa um método diferente e mais rápido para trabalhar com o IntelliSense e exibe uma mensagem para esse efeito na guia **IntelliSense**.

Execute o programa

Agora que o [matplotlib](#) está instalado, execute o programa com (F5) ou sem o depurador (Ctrl+F5) para ver a saída:



Próxima etapa

Aprofunde-se um pouco mais

- Ambientes do Python
- Saiba mais sobre Django no Visual Studio
- Conheça o Flask no Visual Studio

Etapa 6: Trabalhar com o Git

Artigo • 13/01/2024

Etapa anterior: [instalar pacotes e gerenciar o ambiente do Python](#)

Adicionar o controle do código-fonte do Git

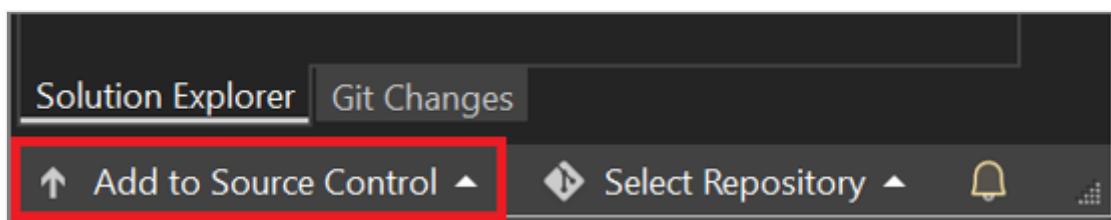
Agora que você criou um aplicativo, pode ser interessante adicioná-lo a um repositório Git. O Visual Studio facilita esse processo com as ferramentas de Git que você pode usar diretamente no IDE.

Dica

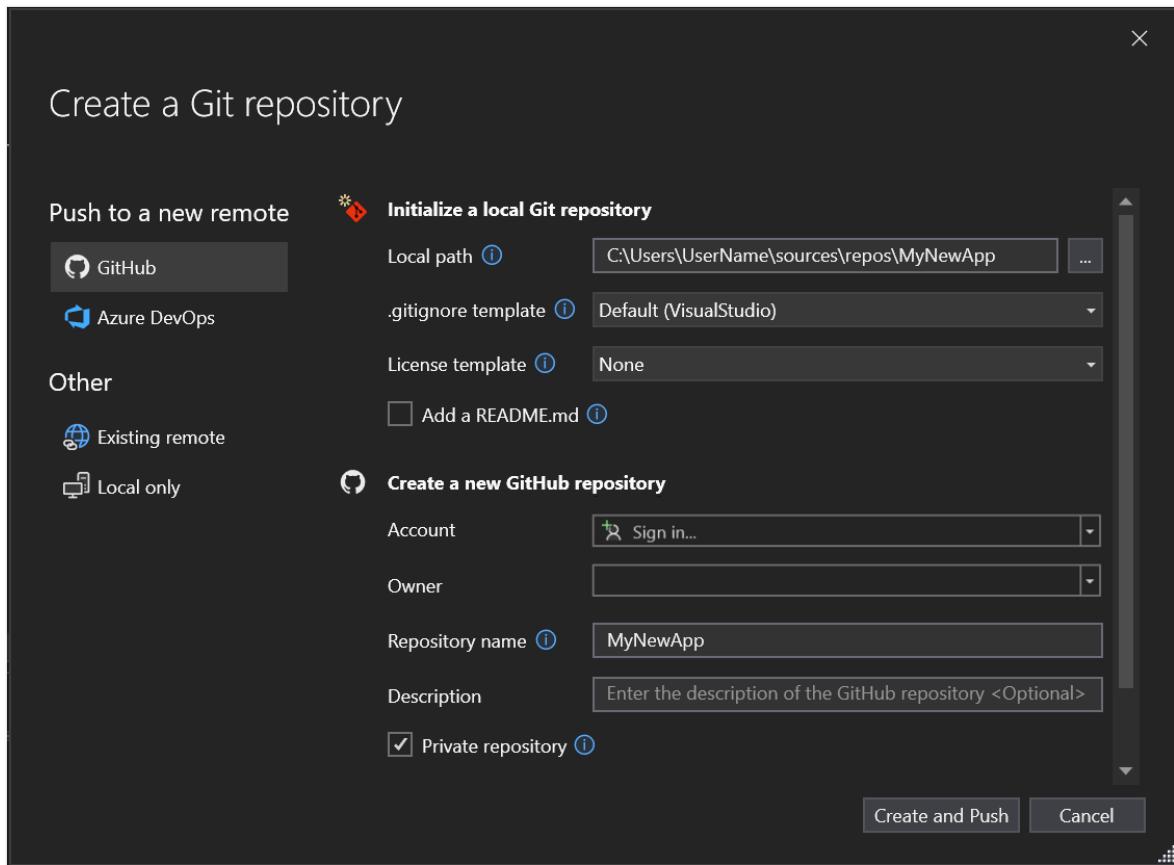
O Git é o sistema de controle de versão moderno mais usado, portanto, se você é um desenvolvedor profissional ou está aprendendo a codificar, o Git pode ser muito útil. Se você é novo no Git, o site <https://git-scm.com/> é um bom local para começar. Lá você vai encontrar roteiros, um livro online popular e vídeos de Conceitos Básicos do Git.

Para associar seu código ao Git, comece criando um repositório Git no local em que o código está localizado:

1. Na barra de status no canto inferior direito do Visual Studio, selecione **Adicionar ao Controle do Código-Fonte** e selecione **Git**.



2. Na caixa de diálogo **Criar um repositório Git**, entre no GitHub.



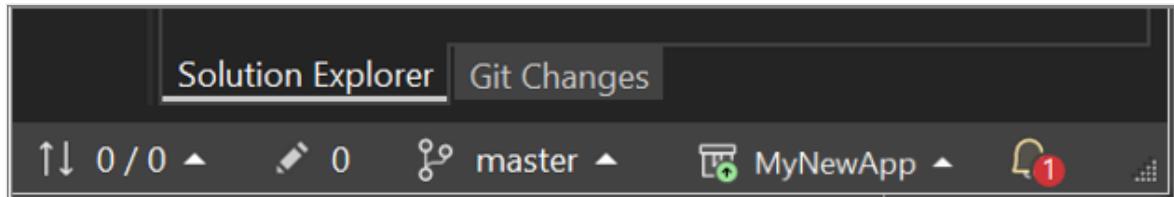
O nome do repositório é preenchido automaticamente com base no local da sua pasta. O novo repositório é privado por padrão, o que significa que você é o único que pode acessá-lo.

Dica

Não importa se o repositório é público ou privado, é melhor ter um backup remoto do código armazenado com segurança no GitHub. Mesmo que você não esteja trabalhando com uma equipe, um repositório remoto disponibiliza seu código para você em qualquer computador.

3. Selecione Criar e Efetuar Push.

Depois de criar o repositório, você verá detalhes do status na barra de status.



O primeiro ícone com as setas mostra quantos commits de saída/entrada estão no branch atual. Você pode usar esse ícone para efetuar pull de qualquer commit de entrada ou efetuar push de commits de saída. Você também pode optar por exibir

primeiro esses commits. Para fazer isso, selecione o ícone e selecione **Exibir Saída/Entrada**.

O segundo ícone com o lápis mostra o número de alterações não confirmadas no código. Você pode selecionar este ícone para exibir essas alterações na janela **Alterações do Git**.

Para saber mais sobre como usar o Git com seu aplicativo, veja a [documentação de controle de versão do Visual Studio](#).

Análise do tutorial

Parabéns por concluir este tutorial sobre Python no Visual Studio. Neste tutorial, você aprendeu como:

- Criar projetos e exibir o conteúdo do projeto.
- Use o editor de código e executar um projeto.
- Use a janela **Interativa** para desenvolver um novo código e copiar facilmente esse código para o editor.
- Executar um programa concluído no depurador do Visual Studio.
- Instalar pacotes e gerenciar ambientes do Python.
- Trabalhar com o código em um repositório do Git.

Agora, explore os Conceitos e guias de Instruções, incluindo os seguintes artigos:

- [Criar uma extensão do C++ para o Python](#)
- [Criação de perfil](#)
- [Teste de unidade](#)

Tutorial: Introdução à estrutura da Web do Django no Visual Studio

Artigo • 08/09/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

O [Django](#) é uma estrutura do Python de alto nível projetada para um desenvolvimento da Web rápido, seguro e escalonável. Este tutorial explora a estrutura do Django no contexto dos modelos de projeto. O Visual Studio fornece os modelos de projeto para simplificar a criação de aplicativos Web baseados no Django.

Neste tutorial, você aprenderá como:

- Criar um projeto básico do Django em um repositório Git usando o modelo "Projeto Web em Branco do Django" (etapa 1).
- Criar um aplicativo do Django com uma página e renderizar essa página usando um modelo (etapa 2).
- Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo (etapa 3).
- Usar o modelo de projeto Web do Django para criar um aplicativo com várias páginas e um design responsivo (etapa 4).
- Autenticar usuários (etapa 5).

Pré-requisitos

- Visual Studio 2022 no Windows com as seguintes opções:
 - A carga de trabalho **desenvolvimento do Python** (guia **Carga de Trabalho** no instalador). Para obter mais instruções, confira [Instalar o suporte do Python no Visual Studio](#).
 - **Git para Windows** na guia **Componentes individuais** em **Ferramentas de código**.

Os modelos de projeto do Django também incluem versões anteriores do plug-in Ferramentas Python para Visual Studio. Os detalhes do modelo podem ser diferentes do que é discutido neste tutorial (especialmente diferente das versões anteriores da estrutura do Django).

No momento, não há suporte para o desenvolvimento do Python no Visual Studio para Mac. No Mac e no Linux, use a [Extensão Python no Visual Studio Code](#).

"Projetos do Visual Studio" e "Projetos do Django"

Na terminologia do Django, um "projeto do Django" tem vários arquivos de configuração no nível do site, juntamente com um ou mais "aplicativos". Para criar um aplicativo Web completo, você pode implantar esses aplicativos em um host da Web. Um projeto do Django pode conter vários aplicativos e o mesmo aplicativo pode estar presente em vários projetos do Django.

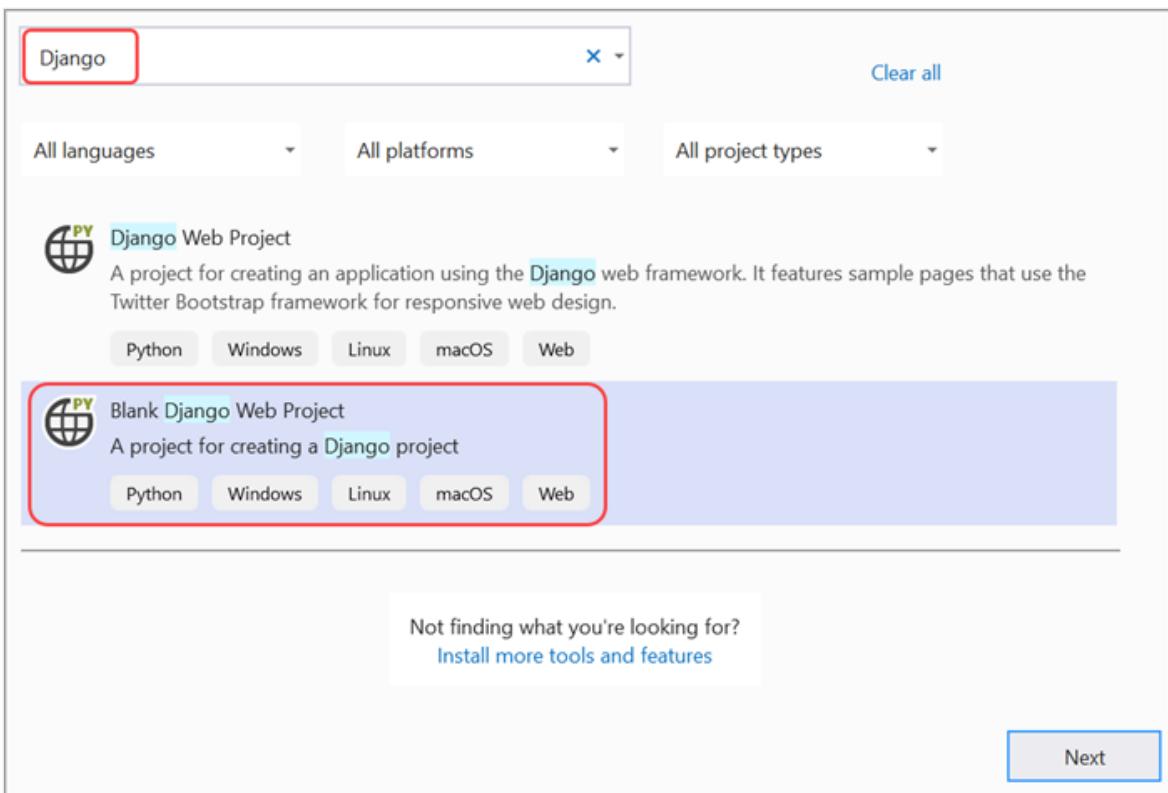
Um projeto do Visual Studio pode conter o projeto do Django juntamente com vários aplicativos. Sempre que este tutorial se referir apenas a um "projeto," ele estará se referindo ao projeto do Visual Studio. Quando se referir ao "projeto do Django" do aplicativo Web, ele estará se referindo ao "projeto do Django" especificamente.

No decorrer deste tutorial, você criará uma única solução do Visual Studio que contém três projetos independentes do Django. Cada projeto contém um único aplicativo do Django. Você pode alternar facilmente entre arquivos diferentes para comparação, mantendo os projetos na mesma solução.

Etapa 1-1: Criar uma solução e um projeto do Visual Studio

Ao trabalhar com o Django na linha de comando, você geralmente inicia um projeto executando o comando `django-admin startproject <project_name>`. No Visual Studio, o modelo "Projeto Web em Branco do Django", fornecerá a mesma estrutura em uma solução e um projeto do Visual Studio.

1. No Visual Studio, escolha **Arquivo > Novo > Projeto**, pesquise "Django" e selecione o modelo **Projeto Web Django em Branco** e, em seguida, selecione **Avançar**.



2. Insira as seguintes informações e selecione **Criar**:

- **Nome do Projeto:** defina o nome do projeto do Visual Studio como **BasicProject**. Esse nome também é usado para o projeto do Django.
- **Local:** especifique um local para criar o projeto e a solução do Visual Studio.
- **Solução:** mantenha esse controle definido com a opção padrão **Criar nova solução**.
- **Nome da solução:** definido como **LearningDjango**, que é apropriado para a solução de um contêiner para vários projetos neste tutorial.

Etapa 1-2: Examinar os controles do Git e publicar em um repositório remoto

Nesta etapa, você se familiarizará com os controles do Git do Visual Studio e o **Team Explorer**. Com a janela do **Team Explorer**, você trabalhará com o controle do código-fonte.

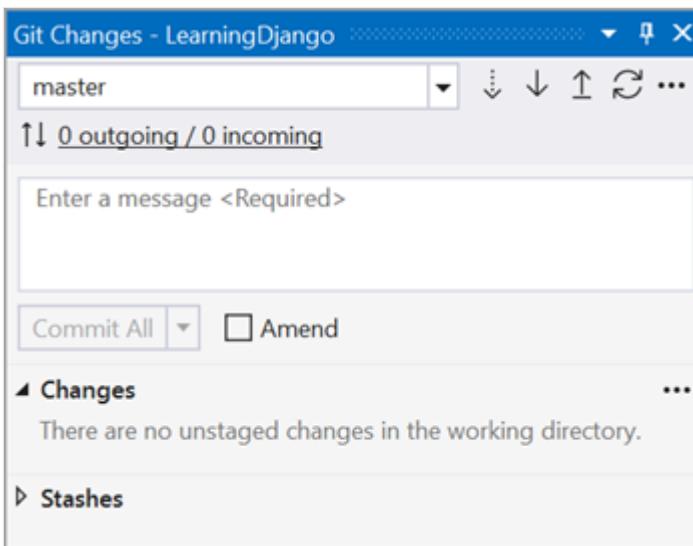
1. Para fazer commit do projeto no controle do código-fonte local:
 - a. Selecione o comando **Adicionar ao Controle do Código-Fonte** no canto inferior da janela principal do Visual Studio.
 - b. Em seguida, selecione a opção **Git**.
 - c. Agora, você será levado para a janela **Criar repositório Git**, na qual poderá criar e enviar por push um novo repositório.

 Add to Source Control

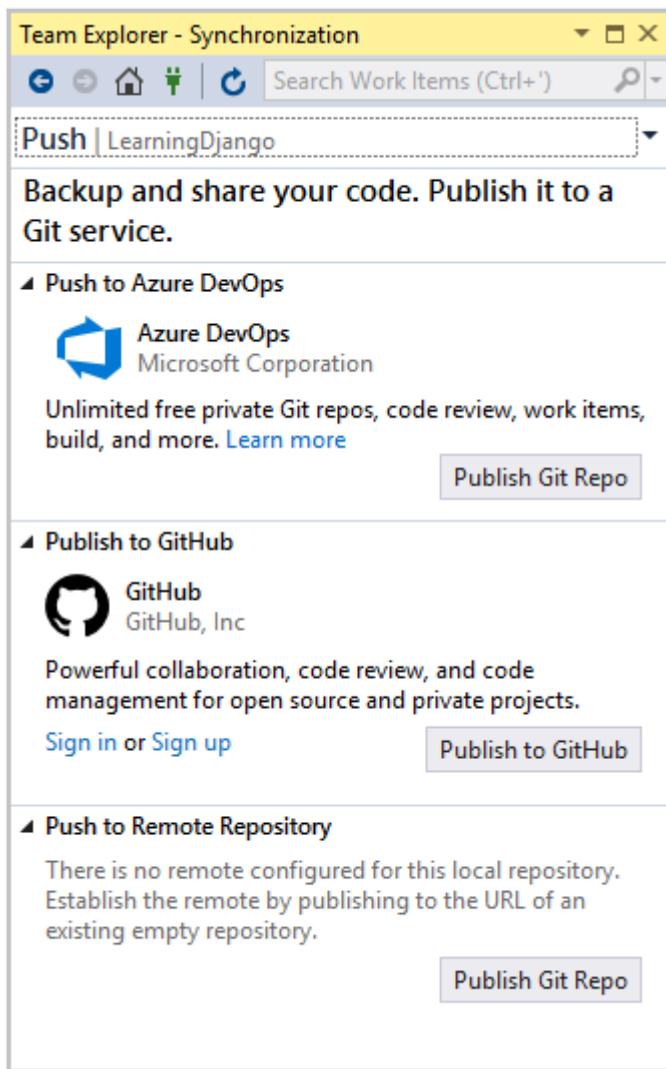
2. Depois de criar um repositório, um conjunto de novos controles do Git aparecerá na parte inferior. Da esquerda para direita, esses controles mostram commits não enviados, alterações não confirmadas, o branch atual e o nome do repositório.



3. Selecione o botão **Alterações do Git**. Em seguida, o Visual Studio abre a página **Alterações do Git no Team Explorer**. Você não vê nenhuma alteração pendente, pois já foi feito commit do projeto recém-criado automaticamente no controle do código-fonte.

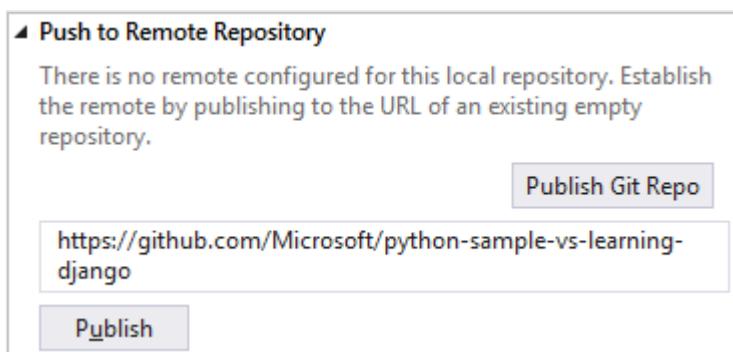


4. Na barra de status do Visual Studio, marque o botão de confirmação não enviada (a seta para cima com um 2) para abrir a página **Sincronização no Team Explorer**. A página **Sincronização** fornece opções fáceis para publicar o repositório local em outros repositórios remotos.



Você pode escolher o serviço que você quiser para seus projetos. Este tutorial mostra o uso do GitHub, em que o código de exemplo concluído do tutorial é mantido no repositório [Microsoft/python-sample-vs-learning-django](https://github.com/Microsoft/python-sample-vs-learning-django).

5. Ao selecionar qualquer um dos controles **Publicar**, o **Team Explorer** solicitará mais informações. Por exemplo, ao publicar o exemplo deste tutorial, o próprio repositório teve que ser criado primeiro, e a opção **Enviar por Push para o Re却itório Remoto** foi usada com a URL do repositório.



Se você não tiver um repositório, as opções **Publicar no GitHub** e **Enviar por Push para o Azure DevOps** permitirão criar um repositório diretamente no Visual Studio.

6. Ao trabalhar com este tutorial, adquira o hábito de usar periodicamente os controles no Visual Studio para confirmar e enviar alterações por push. Este tutorial envia-lhe lembretes nos pontos apropriados.

💡 Dica

Para navegar rapidamente no **Team Explorer**, selecione o cabeçalho (que indica **Alterações** ou **Efetuar Push** nas imagens acima) para ver um menu pop-up das páginas disponíveis.

Pergunta: Quais são algumas vantagens de usar o controle do código-fonte a partir do início de um projeto?

Reposta: o uso do controle do código-fonte desde o início, especialmente se você também usa um repositório remoto, fornece um backup regular do projeto em um local externo. Em vez de manter um projeto em um sistema de arquivos local, o controle do código-fonte oferece um histórico de alterações completo e facilita a reversão de um único arquivo ou todo o projeto para o estado anterior. O histórico de alterações ajuda a determinar a causa das regressões (falhas de teste). Quando várias pessoas estão trabalhando em um projeto, o controle do código-fonte gerencia a substituição e fornece resolução de conflitos.

Por fim, o controle do código-fonte, que é basicamente uma forma de automação, deixa você preparado para automatizar o gerenciamento de builds, testes e versões. É a primeira etapa no uso do DevOps em um projeto. Na verdade, não há razão para não usar o controle do código-fonte desde o início, pois as barreiras à sua adoção são poucas.

Para uma discussão mais aprofundada sobre o controle do código-fonte usado como automação, confira [A fonte da verdade: a função dos repositórios no DevOps](#), um artigo da MSDN Magazine destinado a aplicativos móveis, mas que também se aplica a aplicativos Web.

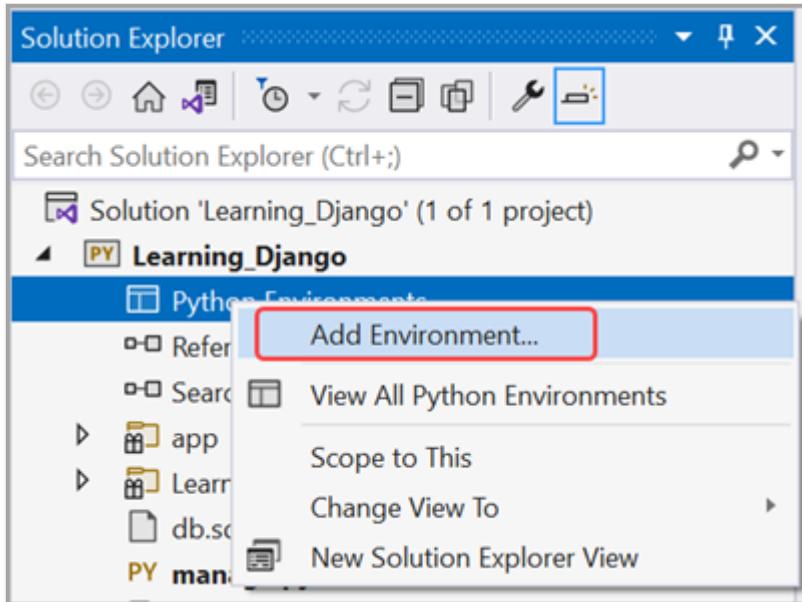
Pergunta: É possível evitar que o Visual Studio faça commit automaticamente de um novo projeto?

Resposta: Sim. Para desabilitar o commit automático, acesse a página **Configurações** no **Team Explorer**. Selecione **Git>Configurações globais**, desmarque a opção rotulada **Confirmar alterações após a mesclagem por padrão** e selecione **Atualizar**.

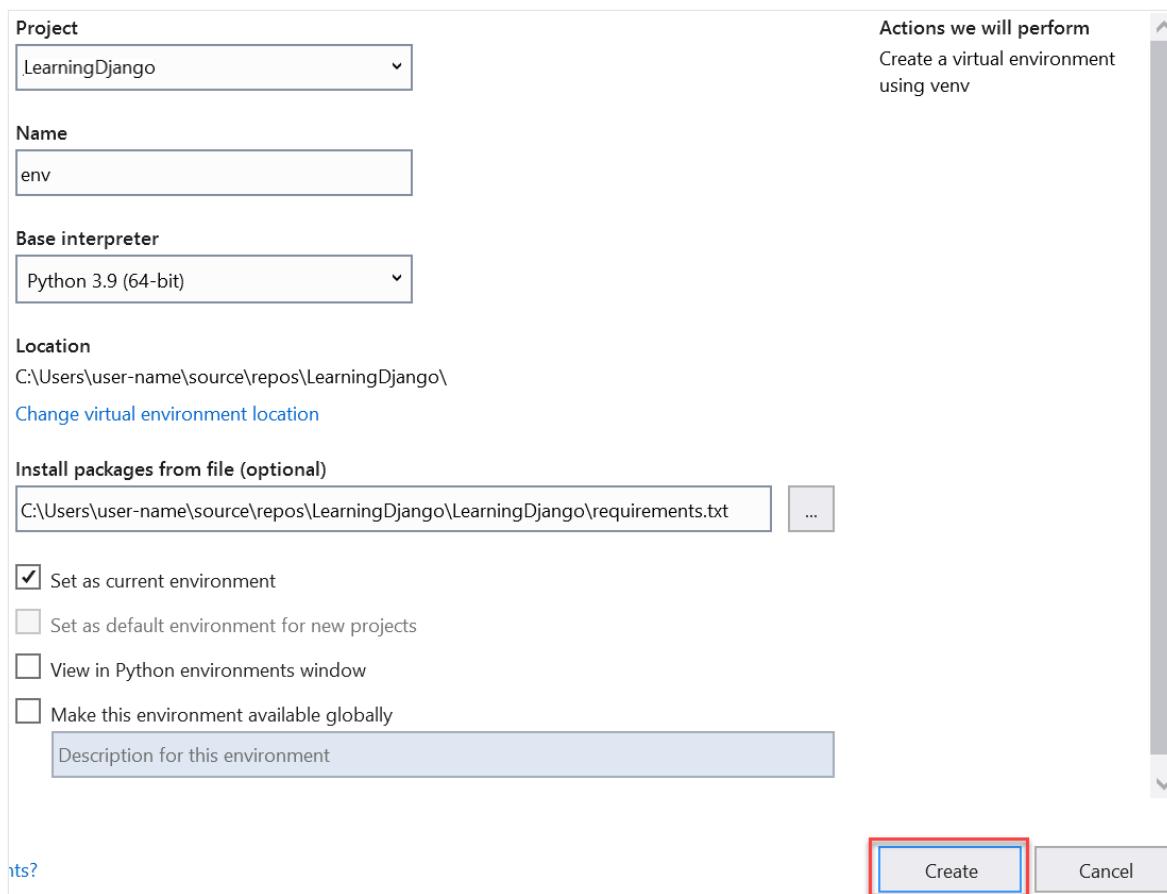
Etapa 1-3: Criar o ambiente virtual e excluí-lo do controle do código-fonte

Agora que você configurou o controle do código-fonte para o projeto, é possível criar o ambiente virtual que contém os pacotes necessários do Django para o projeto. Você pode usar o **Team Explorer** para excluir a pasta do ambiente do controle do código-fonte.

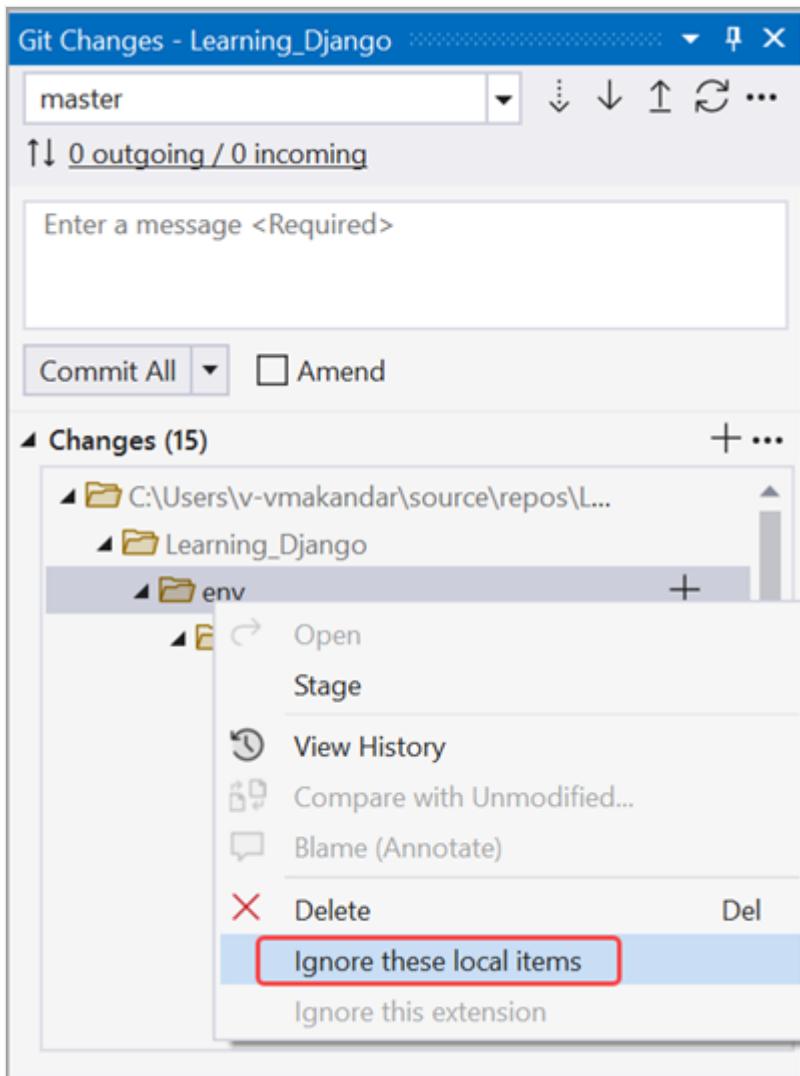
1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Ambientes do Python** e selecione **Adicionar Ambiente**.



2. Selecione **Criar** para aceitar os padrões na caixa de diálogo Adicionar Ambiente Virtual. (Se desejar, o nome do ambiente virtual pode ser alterado. Essa ação alterará apenas o nome da subpasta, mas `env` é uma convenção padrão).



3. Dê a concessão aos privilégios de administrador, se solicitado, e aguarde alguns minutos enquanto o Visual Studio baixa e instala os pacotes. Durante esse tempo, milhares de arquivos são transferidos para muitas subpastas. Confira o progresso na janela **Saída** no Visual Studio. Enquanto você aguarda, analise as seções de perguntas a seguir.
4. Nos controles do Git do Visual Studio (na barra de status), selecione o indicador de alterações (que mostra **99***) que abre a página **Alterações** no **Team Explorer**.
A criação do ambiente virtual gerou milhares de alterações, mas nenhuma delas precisará ser incluída no controle do código-fonte já que você (ou qualquer outra pessoa que venha a clonar o projeto) poderá sempre recriar o ambiente com base no *requirements.txt*.
5. Para excluir o ambiente virtual, clique com o botão direito do mouse na pasta **env** e selecione **Ignorar estes itens locais**.



6. Depois de excluir o ambiente virtual, as únicas alterações restantes são as referentes ao arquivo do projeto e ao arquivo `.gitignore`. O arquivo `.gitignore` contém uma entrada adicional para a pasta do ambiente virtual. Você pode clicar duas vezes no arquivo para ver uma comparação.
7. Digite uma mensagem de confirmação, escolha o botão **Confirmar Todos** e envie as confirmações por push para o repositório remoto.

Pergunta: Por que criar um ambiente virtual?

Resposta: Um ambiente virtual é uma ótima maneira de isolar as dependências exatas do seu aplicativo. Esse isolamento evita conflitos em um ambiente global do Python e auxilia nos testes e na colaboração. Com o tempo, à medida que desenvolver um aplicativo, invariavelmente, você introduzirá muitos pacotes úteis do Python. Você pode atualizar facilmente o arquivo `requirements.txt` do projeto mantendo pacotes em um ambiente virtual específico do projeto. O arquivo `requirements.txt` descreve o ambiente, que está incluído no controle do código-fonte. Quando o projeto é copiado para outros computadores, incluindo servidores de build, servidores de implantação e outros computadores de desenvolvimento, é fácil recriar o ambiente usando apenas o

requirements.txt (é por isso que o ambiente não precisa estar no controle do código-fonte). Para obter mais informações, confira [Usar ambientes virtuais](#).

Pergunta: Como faço para remover um ambiente virtual que já está confirmado no controle do código-fonte?

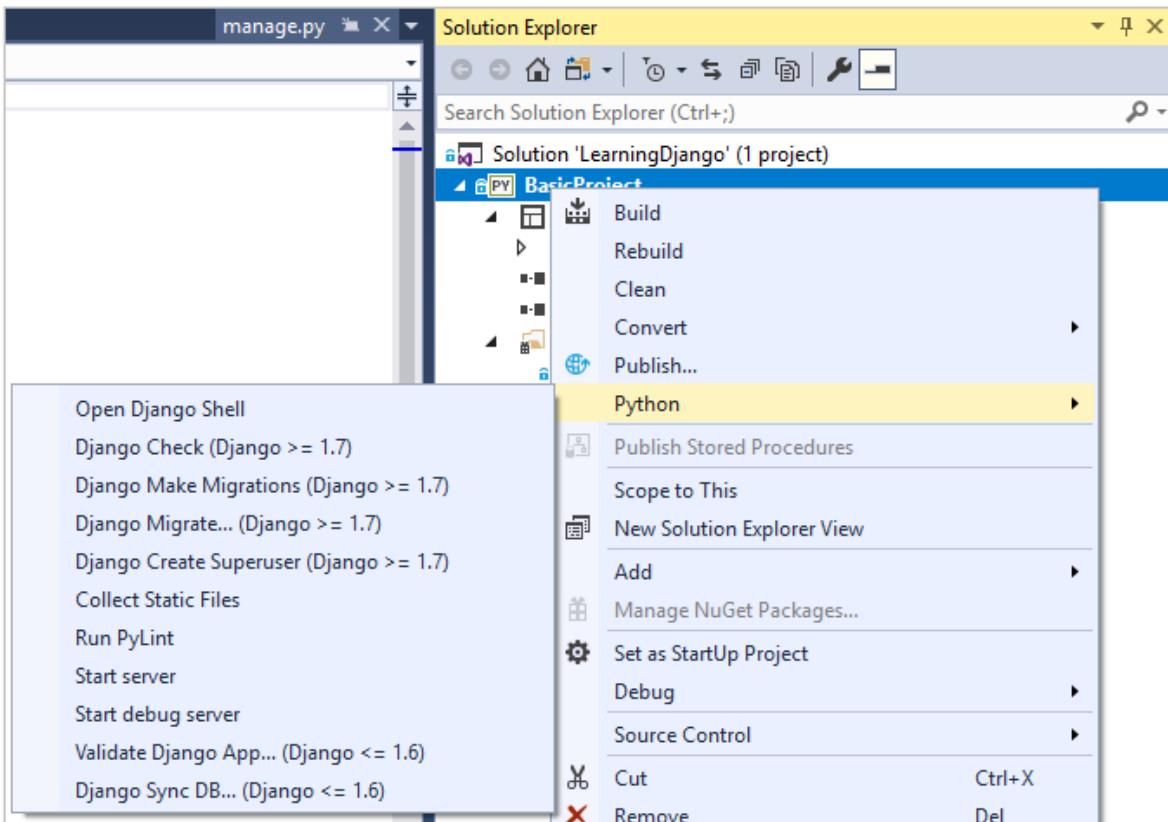
Resposta: primeiro, edite o arquivo `.gitignore` para excluir a pasta. Localize a seção final que traz o comentário `# Python Tools for Visual Studio (PTVS)` e adicione uma nova linha para a pasta do ambiente virtual, como `/BasicProject/env`. (O Visual Studio não mostra o arquivo no **Gerenciador de Soluções**. Para abrir o arquivo diretamente, acesse **Arquivo>Abrir>Arquivo**. Você também pode abrir o arquivo no **Team Explorer**. Acesse a página **Configurações** e selecione **Configurações do Repositório**. Agora, navegue até a seção **Ignorar & Arquivos de Atributos** e selecione o link **Editar** ao lado de `.gitignore`).

Em segundo lugar, abra uma janela de comando, navegue até uma pasta como a *BasicProject*. A pasta *BasicProject* contém a pasta do ambiente virtual, como *env*, e execute `git rm -r env`. Em seguida, confirme essas alterações na linha de comando (`git commit -m 'Remove venv'`) ou confirme na página **Alterações** do **Team Explorer**.

Etapa 1-4: Examinar o código de texto clichê

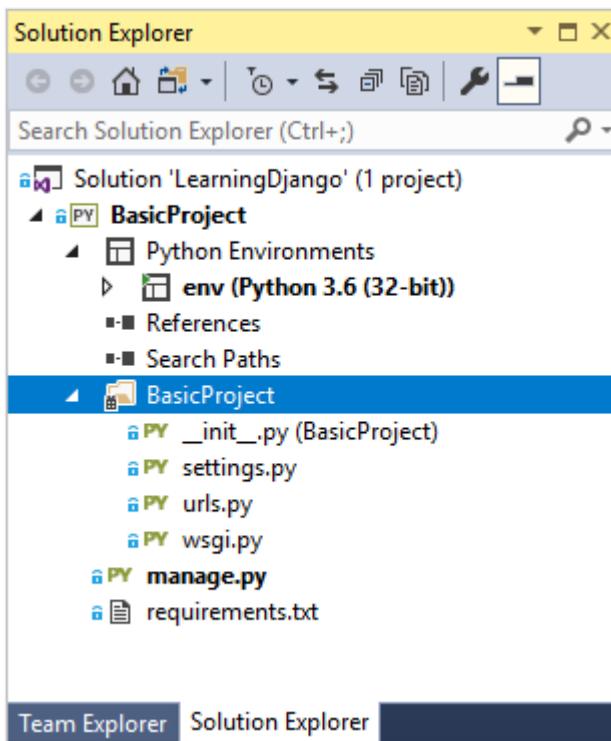
Após concluir a criação do projeto, analise o código do projeto do Django de texto clichê (que é novamente o mesmo gerado pelo comando `django-admin startproject <project_name>` da CLI).

1. A raiz do projeto tem o *manage.py*, o utilitário administrativo de linha de comando do Django que o Visual Studio define automaticamente como o arquivo de inicialização do projeto. Execute o utilitário na linha de comando usando `python manage.py <command> [options]`. Para tarefas comuns do Django, o Visual Studio fornece comandos de menu apropriados. Clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e escolha **Python** para ver a lista. Você vai se deparar com alguns desses comandos no decorrer deste tutorial.



2. Em seu projeto, há uma pasta com o mesmo nome do projeto. Ela contém os arquivos do projeto básico do Django:

- `__init__.py`: um arquivo vazio que informa o Python se essa pasta é um pacote do Python.
- `settings.py`: contém as configurações do projeto do Django, que você vai modificar no decorrer do desenvolvimento de um aplicativo Web.
- `urls.py`: contém um sumário do projeto do Django, que você também vai modificar no decorrer do desenvolvimento.
- `wsgi.py`: um ponto de entrada para os servidores Web compatíveis com WSGI para atender ao projeto. Este arquivo normalmente fica no estado em que se encontra, pois ele fornece os ganchos para os servidores Web de produção.



3. Conforme observado anteriormente, o modelo do Visual Studio também adiciona um arquivo *requirements.txt* ao projeto especificando a dependência de pacote do Django. A presença desse arquivo é que faz com que você seja convidado a criar um ambiente virtual ao desenvolver o projeto pela primeira vez.

Pergunta: O Visual Studio pode gerar um arquivo *requirements.txt* a partir de um ambiente virtual depois de instalar outros pacotes?

Resposta: Sim. Expanda o nó **Ambientes do Python**, clique com o botão direito do mouse no ambiente virtual e escolha o comando **Gerar requirements.txt**. É recomendável usar esse comando periodicamente conforme você modifica o ambiente e confirma as alterações em *requirements.txt* no controle do código-fonte, juntamente com outras alterações de código que dependem desse ambiente. Se você configurar a integração contínua em um servidor de compilação, deverá gerar o arquivo e confirmar as alterações sempre que modificar o ambiente.

Etapa 1-5: Executar o projeto vazio do Django

1. No Visual Studio, selecione **Depurar>Iniciar Depuração (F5)** ou use o botão **Servidor Web** na barra de ferramentas (o navegador que você vai ver pode variar):



2. Executar o servidor significa executar o comando `manage.py runserver <port>`, que inicia o servidor de desenvolvimento interno do Django. Se o Visual Studio informar **Falha ao iniciar o depurador** com uma mensagem que alerta para a ausência de um arquivo de inicialização, clique com o botão direito do mouse em **manage.py** no **Gerenciador de Soluções** e selecione **Definir como Arquivo de Inicialização**.

3. Ao iniciar o servidor, você vê uma janela do console aberta que também exibe o log do servidor. O Visual Studio abre automaticamente um navegador com a página `http://localhost:<port>`. Como o projeto do Django não tem aplicativos, o Django mostra apenas uma página padrão para confirmar que o que você tem até agora está funcionando corretamente.

It worked!

Congratulations on your first Django-powered page.

Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

4. Quando terminar, interrompa o servidor fechando a janela do console ou usando o comando **Depurar>Parar Depuração** no Visual Studio.

Pergunta: O Django é um servidor Web e uma estrutura?

Resposta: Sim e não. O Django tem um servidor Web interno que é usado para fins de desenvolvimento. Esse servidor Web é usado quando você executa o aplicativo Web localmente, por exemplo, durante a depuração no Visual Studio. No entanto, quando você implanta em um host da Web, o Django usa o servidor Web do host. O módulo `wsgi.py` no projeto do Django cuida da conexão com os servidores de produção.

Pergunta: Qual é a diferença entre usar os comandos do menu Depuração e os comandos do servidor no submenu do Python do projeto?

Resposta: Além dos comandos do menu **Depurar** e dos botões da barra de ferramentas, você também pode iniciar o servidor usando os comandos **Python>Executar servidor** ou **Python>Executar o servidor de depuração** no menu de contexto do projeto. Os dois comandos abrem uma janela de console na qual você vai ver a URL local (`localhost:port`) do servidor em execução. No entanto, você deve abrir manualmente um navegador usando essa URL já que a execução do servidor de depuração não inicia

automaticamente o depurador do Visual Studio. Se desejar, você pode anexar um depurador ao processo em execução usando o comando **Depurar>Anexar ao Processo**.

Próximas etapas

Neste ponto, o projeto básico do Django não tem aplicativos. Você vai criar um aplicativo na próxima etapa. Como você vai trabalhar mais com aplicativos do Django do que com o projeto do Django, não será necessário saber detalhes dos arquivos boilerplate por enquanto.

[Criar um aplicativo do Django com modos de exibição e modelos de página](#)

Aprofunde-se um pouco mais

- Código do projeto do Django: [Como gravar seu primeiro aplicativo do Django, parte 1 ↗](#) (docs.djangoproject.com)
- Utilitário administrativo: [django-admin e manage.py ↗](#) (docs.djangoproject.com)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-django ↗](#)

Etapa 2: Criar um aplicativo Django com exibição e modelos de página

Artigo • 08/09/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Criar uma solução e um projeto do Visual Studio](#)

No projeto do Visual Studio, você só tem os componentes do nível do site de um *projeto* do Django agora, que pode executar um ou mais *aplicativos* de Django. A próxima etapa é criar seu primeiro aplicativo com uma única página.

Nesta etapa, você aprenderá a:

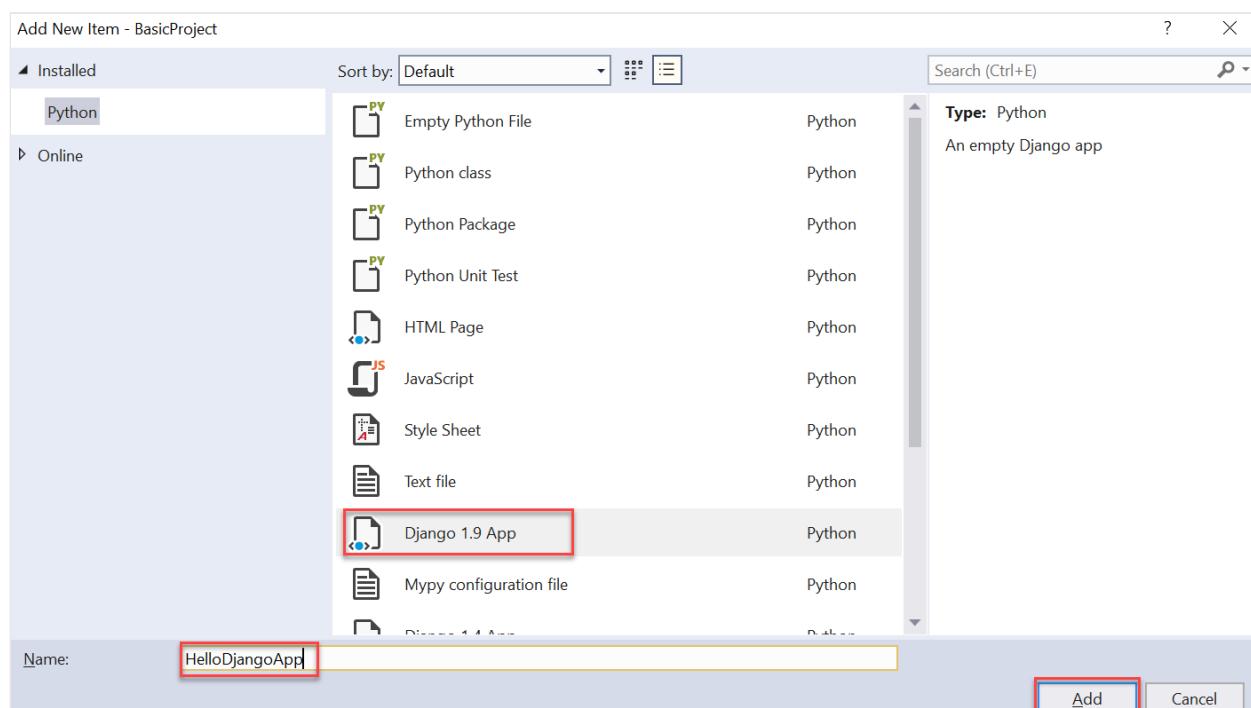
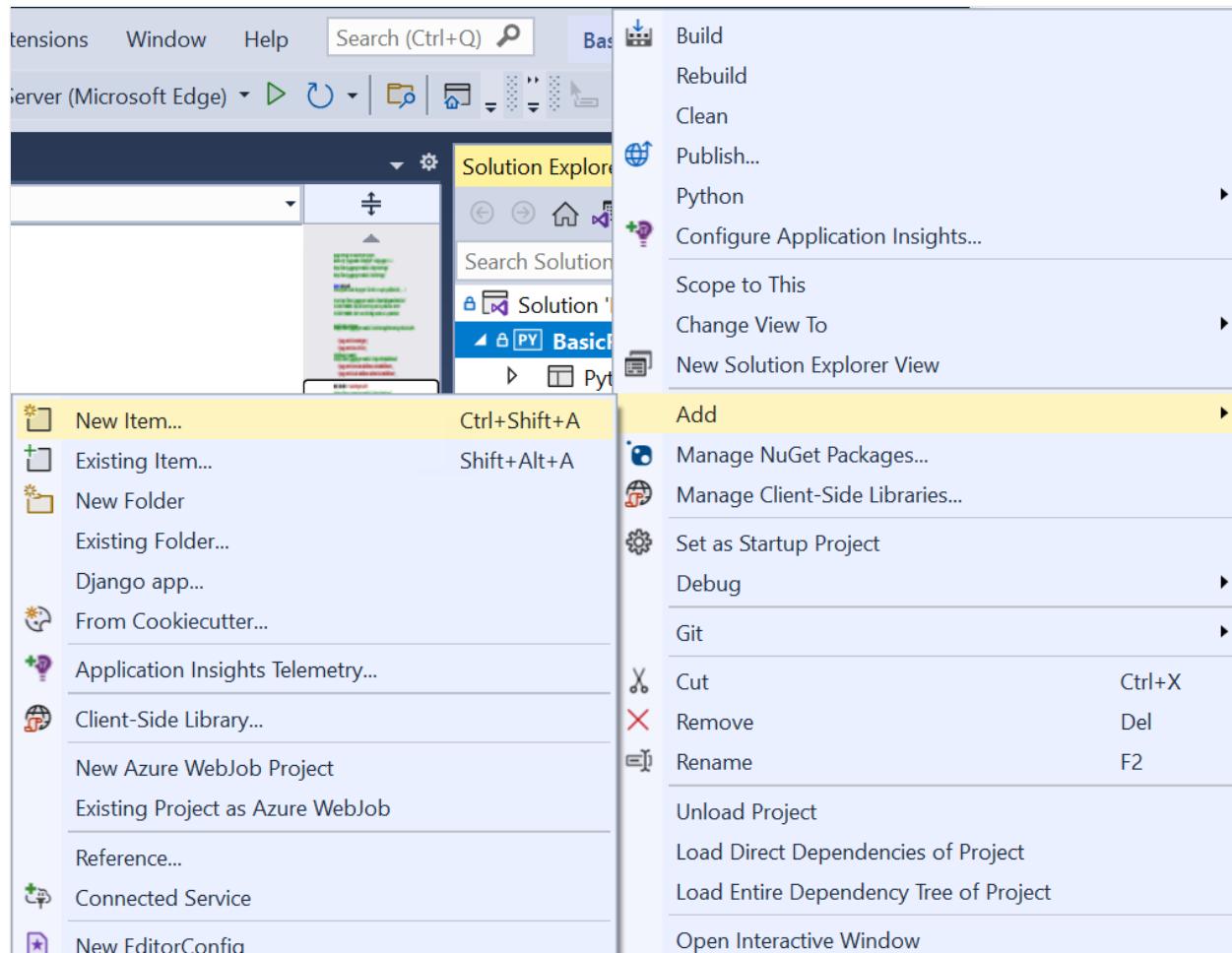
- ✓ Criar um aplicativo do Django com uma única página (etapa 2-1)
- ✓ Executar o aplicativo do projeto do Django (etapa 2-2)
- ✓ Renderizar um modo de exibição usando HTML (etapa 2-3)
- ✓ Renderizar um modo de exibição usando um modelo de página do Django (etapa 2-4)

Etapa 2-1: Criar um aplicativo com uma estrutura padrão

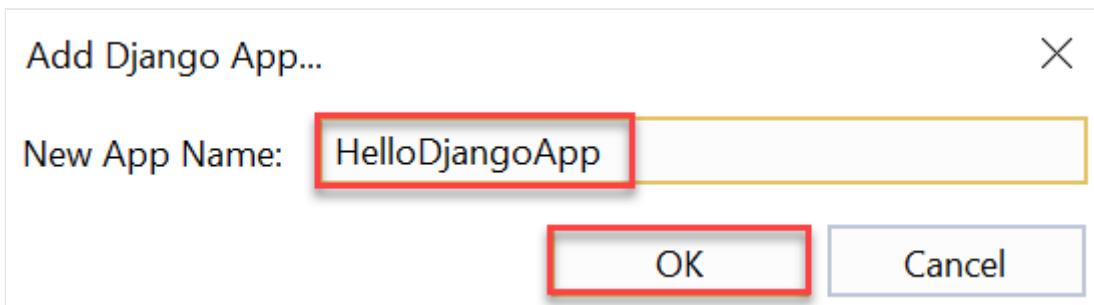
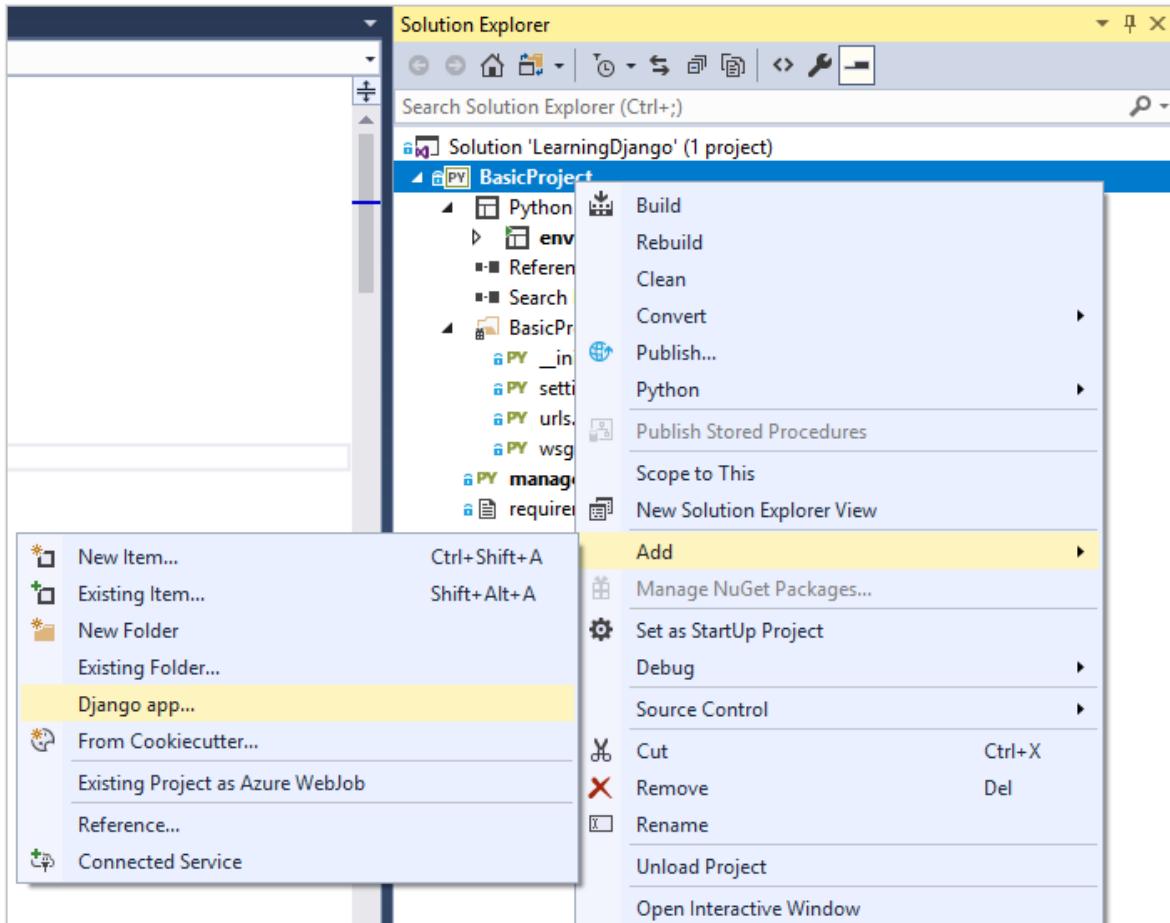
Um aplicativo do Django é um pacote separado em Python que contém um conjunto de arquivos relacionados para uma finalidade específica. Um projeto do Django pode conter muitos aplicativos, que ajudam um host da Web a fornecer muitos pontos de entrada separados de um único nome de domínio. Por exemplo, um projeto do Django para um domínio como `contoso.com` pode conter um aplicativo para `www.contoso.com`, um segundo aplicativo para `support.contoso.com` e um aplicativo de terceiro para `docs.contoso.com`. Nesse caso, o projeto do Django manipula o roteamento e as configurações de URL no nível do site (em seus arquivos `urls.py` e `settings.py`). Cada aplicativo tem seu próprio estilo e comportamento por meio do seu roteamento interno, exibições, modelos, arquivos estáticos e interface administrativa.

Um aplicativo do Django normalmente começa com um conjunto padrão de arquivos. O Visual Studio fornece modelos para inicializar um aplicativo do Django dentro de um projeto do Django, junto com um comando de menu integrado que tem a mesma finalidade:

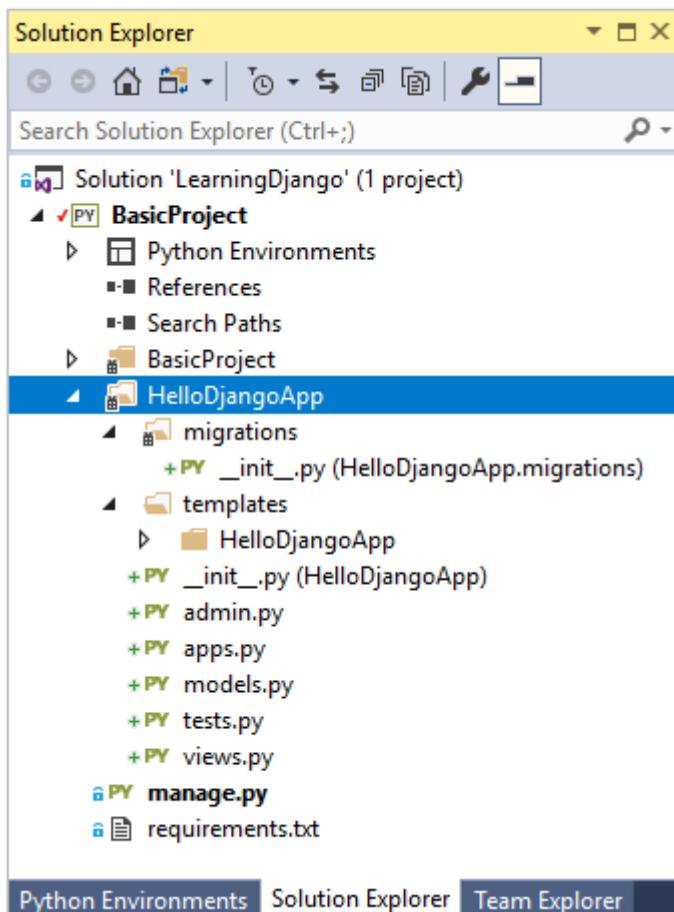
- Modelos: no Gerenciador de Soluções, clique com botão direito do mouse no projeto e selecione **Adicionar>Novo item**. Na caixa de diálogo **Adicionar Novo Item**, selecione o modelo **Aplicativo do Django 1.9**, especifique o nome do aplicativo no campo **Nome** e selecione **Adicionar**.



- Integrado comando: no **Gerenciador de Soluções**, clique com o botão direito no projeto e selecione **Adicionar>Aplicativo do Django**. Esse comando solicita um nome para você. Especifique o nome do aplicativo no campo **Novo Nome de Aplicativo** e selecione **OK**.



Usando qualquer método, crie um aplicativo com o nome "HelloDjangoApp". Agora, a pasta "HelloDjangoApp" é criada em seu projeto. A pasta contém os seguintes itens:



| Item | Descrição |
|--------------------------|---|
| migrações | Uma pasta na qual o Django armazena scripts que atualizam o banco de dados para se alinhar com as alterações nos modelos. Em seguida, as ferramentas de migração do Django aplicam as alterações necessárias a qualquer versão anterior do banco de dados para corresponder aos modelos atuais. Com as migrações, você mantém o foco nos seus modelos e permite que o Django lide com o esquema de banco de dados subjacente. As migrações são discutidas na Documentação do Django . No momento, a pasta contém um arquivo <code>__init__.py</code> (o que indica que a pasta define seu próprio pacote do Python). |
| <code>__init__.py</code> | O arquivo que identifica o aplicativo como um pacote. |
| templates | Uma pasta para modelos de página do Django que contém um único arquivo <code>index.html</code> . O arquivo <code>index.html</code> é colocado na pasta com o mesmo nome do aplicativo. (No Visual Studio 2017 15.7 e versões anteriores, o arquivo é colocado diretamente em <code>modelos</code> e a etapa 2-4 instrui você a criar a subpasta). Modelos são blocos de HTML aos quais os modos de exibição podem adicionar informações para renderizar dinamicamente uma página. "Variáveis" de modelos de página, como <code>{{ content }}</code> em <code>index.html</code> , são espaços reservados para valores dinâmicos, conforme explicado mais adiante neste artigo (etapa 2). Normalmente, os aplicativos do Django criam um namespace para seus modelos, colocando-os em uma subpasta que corresponda ao nome do aplicativo. |
| <code>admin.py</code> | O arquivo Python no qual você estende a interface administrativa do aplicativo, que é usada para exibir e editar dados em um banco de dados. Inicialmente, esse arquivo contém apenas a instrução <code>from django.contrib import admin</code> . Por padrão, o Django |

| Item | Descrição |
|------------------|---|
| | incluir uma interface administrativa padrão por meio de entradas no arquivo <i>settings.py</i> do projeto do Django. Para ativar a interface, você pode remover a marca de comentário das entradas existentes no arquivo <i>urls.py</i> . |
| apps.py | Um arquivo Python que define uma classe de configuração para o aplicativo (veja abaixo, depois desta tabela). |
| models.py | Modelos são objetos de dados, identificados por funções, por meio dos quais os modos de exibição interagem com o banco de dados subjacente do aplicativo. O Django fornece a camada de conexão de banco de dados para que os aplicativos não se preocupem com os detalhes dos modelos. O arquivo <i>models.py</i> é um local padrão onde você cria seus modelos. Inicialmente, o arquivo <i>models.py</i> contém apenas a instrução <code>from django.db import models</code> . |
| tests.py | Um arquivo Python que contém a estrutura básica de testes de unidade. |
| views.py | Os modos de exibição são semelhantes às páginas da Web, que recebem uma solicitação HTTP e retornam uma resposta HTTP. Os modos de exibição costumam ser renderizados como HTML e os navegadores da Web sabem como exibir, mas um modo de exibição não precisa necessariamente ser visível (como um formulário intermediário). Um modo de exibição é definido por uma função Python que tem a responsabilidade de renderizar o HTML no navegador. O arquivo <i>views.py</i> é um local padrão onde você cria suas exibições. Inicialmente, o arquivo <i>views.py</i> contém apenas a instrução <code>from django.shortcuts import render</code> . |

Quando você usa o nome "HelloDjangoApp", o conteúdo do arquivo *apps.py* aparece como:

```
Python

from django.apps import AppConfig

class HelloDjango AppConfig(AppConfig):
    name = 'HelloDjango'
```

Pergunta: Há alguma diferença entre criar um aplicativo do Django no Visual Studio e criar um aplicativo na linha de comando?

Resposta: A execução do comando **Adicionar>Aplicativo do Django** ou o uso de **Adicionar>Novo Item** com um modelo de aplicativo do Django produz os mesmos arquivos que o comando do Django `manage.py startapp <app_name>`. A vantagem de criar um aplicativo no Visual Studio é que a pasta do aplicativo e todos os seus arquivos

são automaticamente integrados ao projeto. Você pode usar o mesmo comando do Visual Studio para criar qualquer número de aplicativos em seu projeto.

Etapa 2-2: Executar o aplicativo a partir do projeto do Django

Neste ponto, se você executar novamente o projeto no Visual Studio (usando o botão de barra de ferramentas ou **Depurar>Iniciar depuração**), você ainda verá a página padrão. Nenhum conteúdo do aplicativo é exibido porque você precisa definir uma página específica do aplicativo e adicionar o aplicativo ao projeto do Django:

1. Na pasta *HelloDjangoApp*, modifique o arquivo *views.py* para definir uma exibição chamada "index":

```
Python

from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, Django!")
```

2. Na pasta *BasicProject* (criada na etapa 1), modifique o arquivo *urls.py* para que ele corresponda ao seguinte código (você poderá manter o comentário instrutivo se desejar):

```
Python

from django.urls import include, re_path
import HelloDjangoApp.views

# Django processes URL patterns in the order they appear in the array
urlpatterns = [
    re_path(r'^$', HelloDjangoApp.views.index, name='index'),
    re_path(r'^home$', HelloDjangoApp.views.index, name='home')
]
```

Cada padrão de URL descreve as exibições para as quais o Django encaminha URLs relativas do site específicas (ou seja, a parte que segue <https://www.domain.com/>). A primeira entrada em `urlPatterns` que começa com a expressão regular `^$` é o roteamento para a raiz do site, `/`. A segunda entrada, `^home$` direcionado especificamente para `/home`. É possível ter vários roteamentos para o mesmo modo de exibição.

3. Execute o projeto novamente para ver a mensagem **Olá, Django!**, conforme definido pelo modo de exibição. Pare o servidor ao terminar.

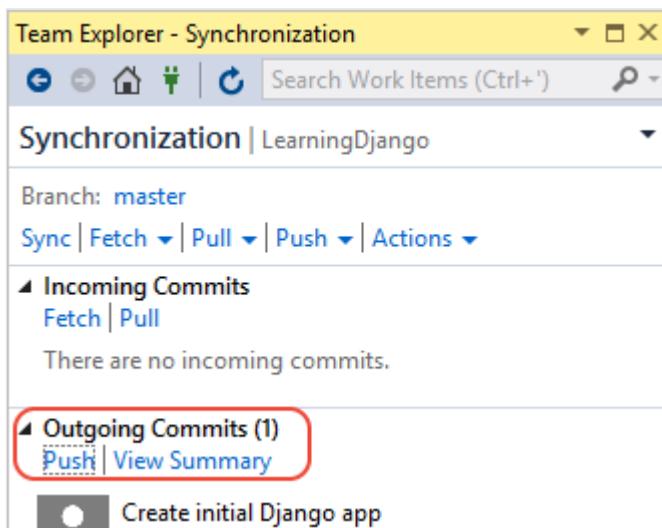
Confirmar o controle do código-fonte

Depois de fazer alterações no código e testar com êxito, você pode examinar e confirmar o controle do código-fonte. Em etapas posteriores, quando este tutorial lembrar você de fazer commit com o controle do código-fonte novamente, consulte esta seção.

1. Selecione o botão de alterações na parte inferior do Visual Studio (circulado abaixo), que encaminha para o **Team Explorer**.



2. No **Team Explorer**, digite uma mensagem de confirmação, como "Criar aplicativo inicial do Django" e selecione **Confirmar Tudo**. Quando a confirmação for concluída, você verá uma mensagem **Commit <hash>** criada localmente. **Sincronize para compartilhar suas alterações com o servidor**. Se você quiser enviar alterações por push para o repositório remoto, selecione **Sincronizar** e, em seguida, selecione **efetuar push** em **Commits de saída**. Também é possível acumular várias confirmações locais antes de enviar para o repositório remoto.



Pergunta: Para que serve o prefixo 'r' antes das cadeias de caracteres de roteamento?

Resposta: O prefixo "r" em uma cadeia de caracteres em Python significa "raw", que instrui o Python a não escapar de nenhum caractere dentro da cadeia. As expressões regulares usam muitos caracteres especiais. O uso do prefixo "r" torna as cadeias de caracteres muito mais fáceis de serem lidas do que os caracteres de escape "\".

Pergunta: O que significam os caracteres ^ e \$ nas entradas de roteamento de URL?

Resposta: Nas expressões regulares que definem padrões de URL, ^ significa "início de linha" e \$ significa "fim de linha", em que as URLs são relativas à raiz do site (a parte que segue `https://www.domain.com/`). A expressão regular `^$` efetivamente significa "em branco" e corresponde à URL completa `https://www.domain.com/` (nada é adicionado à raiz do site). O padrão `^home$` corresponde exatamente a `https://www.domain.com/home/`. (O Django não usa o / à direita na correspondência de padrões).

Se você não usar um \$ à direita em uma expressão regular, assim como em `^home`, o padrão de URL corresponderá a *qualquer* URL que comece com "home", como "home", "homework", "homestead" e "home192837".

Para fazer experiências com expressões regulares diferentes, use ferramentas online, como regex101.com e pythex.org.

Etapa 2-3: Renderizar um modo de exibição usando HTML

A função `index` que você tem no arquivo `views.py` não gera uma resposta HTTP de texto sem formatação para a página. A maioria das páginas da Web reais respondem com páginas HTML avançadas que incorporam dados dinâmicos com frequência. Na verdade, a principal razão para definir uma exibição usando uma função é gerar esse conteúdo dinamicamente.

Como o argumento para `HttpResponse` é apenas uma cadeia de caracteres, você pode criar qualquer HTML que quiser em uma cadeia de caracteres. Como um exemplo simples, substitua a função `index` pelo código a seguir (mantenha as instruções existentes `from`). Em seguida, a função `index` gerará uma resposta HTML usando conteúdo dinâmico que é atualizado sempre que você atualizar a página.

Python

```
from datetime import datetime

def index(request):
    now = datetime.now()

    html_content = "<html><head><title>Hello, Django</title></head><body>"
    html_content += "<strong>Hello Django!</strong> on " + now.strftime("%A,
    %d %B, %Y at %X")
```

```
html_content += "</body></html>"  
  
return HttpResponse(html_content)
```

Agora, execute o projeto novamente para ver uma mensagem como "Olá, Django!" na segunda-feira, 16 de abril de 2018 às 16:28:10". Atualize a página para atualizar a hora e confirme que o conteúdo está sendo gerado com cada solicitação. Pare o servidor ao terminar.

💡 Dica

Um atalho para parar e reiniciar o projeto é usar o comando de menu **Depurar>Reiniciar (Ctrl+Shift+F5)** ou o botão **Reiniciar** na barra de ferramentas de depuração:



Etapa 2-4: Renderizar um modo de exibição usando um modelo de página

A geração de HTML em código funciona bem para páginas pequenas. No entanto, à medida que as páginas ficam mais sofisticadas, você precisa manter as partes HTML estáticas da sua página (juntamente com referências a arquivos CSS e JavaScript) como "modelos de página". Em seguida, você pode inserir o conteúdo dinâmico gerado por código nos modelos de página. Na seção anterior, apenas a data e a hora da chamada `now.strftime` são dinâmicas, o que significa que todos os outros conteúdos podem ser colocados em um modelo de página.

Um modelo de página do Django é um bloco de HTML que contém vários tokens de substituição chamados "variáveis". As variáveis são delineadas por `{{` e `}}`, por exemplo, `{{ content }}`. O módulo de modelo do Django substitui as variáveis por conteúdo dinâmico que você fornece no código.

As etapas a seguir demonstram o uso de modelos de página:

1. Na pasta *BasicProject*, que contém o projeto do Django, abra o arquivo `settings.py`. Adicione o nome do aplicativo, "HelloDjangoApp", à lista `INSTALLED_APPS`. A adição do aplicativo à lista informa ao projeto do Django que há uma pasta com o nome "HelloDjangoApp" contendo um aplicativo:

```
INSTALLED_APPS = [
    'HelloDjangoApp',
    # Other entries...
]
```

2. No arquivo `settings.py`, verifique se o objeto `TEMPLATES` contém a linha a seguir (incluída por padrão). O código a seguir instrui o Django a procurar modelos na pasta de `modelos` de um aplicativo instalado:

JSON

```
'APP_DIRS': True,
```

3. Na pasta `HelloDjangoApp`, abra o arquivo de modelo de página `templates/HelloDjangoApp/index.html` (ou `templates/index.html` no VS 2017 15.7 e anteriores), para observar que ele contém uma variável, a `{{ content }}`:

HTML

```
<html>
  <head>
    <title></title>
  </head>

  <body>
    {{ content }}
  </body>
</html>
```

4. Na pasta `HelloDjangoApp`, abra o arquivo `views.py` e substitua a função `index` pelo código a seguir, que usa a função auxiliar `django.shortcuts.render`. O auxiliar `render` fornece uma interface simplificada para trabalhar com modelos de página. Mantenha todas as instruções `from` existentes.

Python

```
from django.shortcuts import render    # Added for this step

def index(request):
    now = datetime.now()

    return render(
        request,
        "HelloDjangoApp/index.html",  # Relative path from the
        'templates' folder to the template file
        # "index.html", # Use this code for VS 2017 15.7 and earlier
```

```
        {
            'content': "<strong>Hello Django!</strong> on " +
now.strftime("%A, %d %B, %Y at %X")
        }
    )
```

O primeiro argumento para `render` é o objeto de solicitação, seguido do caminho relativo para o arquivo de modelo dentro da pasta `templates` do aplicativo. Um arquivo de modelo recebe o nome do modo de exibição ao qual ele oferece suporte, se apropriado. O terceiro argumento para `render` é um dicionário de variáveis ao qual o modelo se refere. Você pode incluir objetos no dicionário e, nesse caso, uma variável no modelo pode se referir a `{{ object.property }}`.

5. Execute o projeto e observe o resultado. Você deve ver uma mensagem semelhante àquela vista na etapa 2-2, indicando que o modelo funciona.

Observe que o HTML usado na propriedade `content` é processado apenas como texto simples, porque a função `render` escapa automaticamente o HTML. O escape automático impede vulnerabilidades acidentais a ataques de injeção. Os desenvolvedores geralmente coletam entradas de uma página e a usam como um valor em outra por meio de um espaço reservado de modelo. O escape também funciona como um lembrete de que é melhor manter o HTML no modelo da página e fora do código. Além disso, é simples criar mais variáveis quando necessário. Por exemplo, altere o arquivo `index.html` com `modelos` para corresponder à marcação a seguir. A marcação a seguir adiciona um título de página e mantém toda a formatação no modelo de página:

```
HTML
```

```
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <strong>{{ message }}</strong>{{ content }}
  </body>
</html>
```

Depois, para fornecer valores para todas as variáveis no modelo de página, escreva a função do modo de exibição `index` conforme especificado a seguir:

```
Python
```

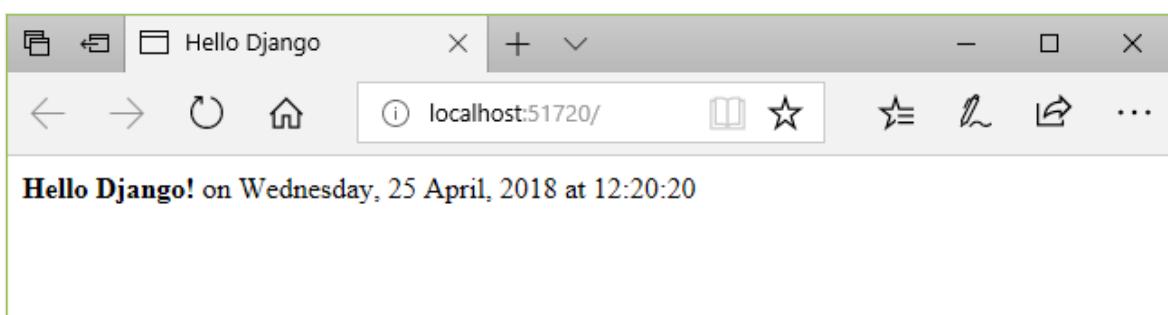
```
def index(request):
    now = datetime.now()
```

```

    return render(
        request,
        "HelloDjangoApp/index.html", # Relative path from the
        'templates' folder to the template file
        # "index.html", # Use this code for VS 2017 15.7 and earlier
        {
            'title' : "Hello Django",
            'message' : "Hello Django!",
            'content' : " on " + now.strftime("%A, %d %B, %Y at %X")
        }
)

```

6. Interrompa o servidor e reinicie o projeto. Verifique se a página é renderizada corretamente:



7. Visual Studio 2017 versão 15.7 e anteriores: como etapa final, mova seus modelos para uma subpasta com o mesmo nome do aplicativo. A subpasta cria um namespace e evita possíveis conflitos com outros aplicativos que você pode adicionar ao projeto. (Os modelos no VS 2017 15.8+ fazem isso automaticamente.) Ou seja, crie uma subpasta em *templates* chamada *HelloDjangoApp*, mova o arquivo *index.html* para essa subpasta e modifique a função de exibição `index`. A função de exibição `index` fará referência ao novo caminho do modelo, *HelloDjangoApp/index.html*. Em seguida, execute o projeto, verifique se a página é renderizada corretamente e pare o servidor.
8. Confirme suas alterações no controle do código-fonte e atualize seu repositório remoto, se necessário, conforme descrito em [etapa 2-2](#).

Pergunta: Os modelos de página precisam ficar em um arquivo separado?

Resposta: normalmente, os modelos são mantidos em arquivos HTML separados. Você também pode usar um modelo embutido. Para manter uma distinção clara entre marcação e código, é recomendável usar um arquivo separado.

Pergunta: Os modelos precisam usar a extensão de arquivo .html?

Resposta: A extensão *.html* para arquivos de modelo de página é totalmente opcional, porque você sempre identifica o caminho relativo exato para o arquivo no segundo argumento para a função `render`. No entanto, o Visual Studio (e outros editores) fornece funcionalidades como preenchimento de código e coloração de sintaxe com arquivos *.html*, o que supera o fato de os modelos de página não serem estritamente HTML.

De fato, quando você está trabalhando com um projeto do Django, o Visual Studio detecta automaticamente o arquivo HTML que possui um modelo do Django e fornece determinados recursos de preenchimento automático. Por exemplo, quando você começa a digitar um comentário de modelo de página do Django, `{#`, o Visual Studio fornece automaticamente os caracteres de fechamento `#}`. Os comandos **Comentar Seleção** e **Remover Marca de Comentário da Seleção** (no menu **Editar>Avançado** e na barra de ferramentas) também usam comentários de modelo em vez de comentários em HTML.

Pergunta: Quando executo o projeto, é exibido um erro indicando que não é possível encontrar o modelo. Qual é o problema?

Resposta: Se forem exibidos erros indicando que o modelo não pode ser encontrado, verifique se você adicionou o aplicativo a `settings.py` do projeto do Django na lista `INSTALLED_APPS`. Sem essa entrada, o Django não saberá o que procurar na pasta `templates` do aplicativo.

Pergunta: Por que o namespace do modelo é importante?

Resposta: Quando o Django procura por um modelo referido na função `render`, ele usa o primeiro arquivo encontrado que corresponda ao caminho relativo. Se você tiver vários aplicativos do Django no mesmo projeto com as mesmas estruturas de pasta para os modelos, é provável que um aplicativo use inadvertidamente um modelo de outro aplicativo. Para evitar esses erros, sempre crie uma subpasta na pasta `templates` do aplicativo que corresponda ao nome do aplicativo para evitar qualquer duplicação.

Próximas etapas

Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo

Aprofunde-se um pouco mais

- [Como gravar seu primeiro aplicativo do Django, parte 1 – \(modos de exibição\)](#) ↗ (docs.djangoproject.com)
- Para conhecer mais recursos de modelos do Django, como os elementos inclui e herança, confira [A linguagem de modelos do Django](#) ↗ (docs.djangoproject.com)
- [Treinamento em expressões regulares no inLearning](#) ↗ (LinkedIn)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-django](#) ↗

Etapa 3: Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo com o aplicativo Django

Artigo • 18/03/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Criar um aplicativo do Django com modos de exibição e modelos de página](#)

Nas etapas anteriores deste tutorial, você aprendeu como criar um aplicativo mínimo em Django com uma única página HTML. Os aplicativos Web modernos, no entanto, contêm muitas páginas. As páginas dos aplicativos web modernos usam recursos compartilhados, como arquivos CSS e JavaScript, para fornecer comportamento e estilo consistentes.

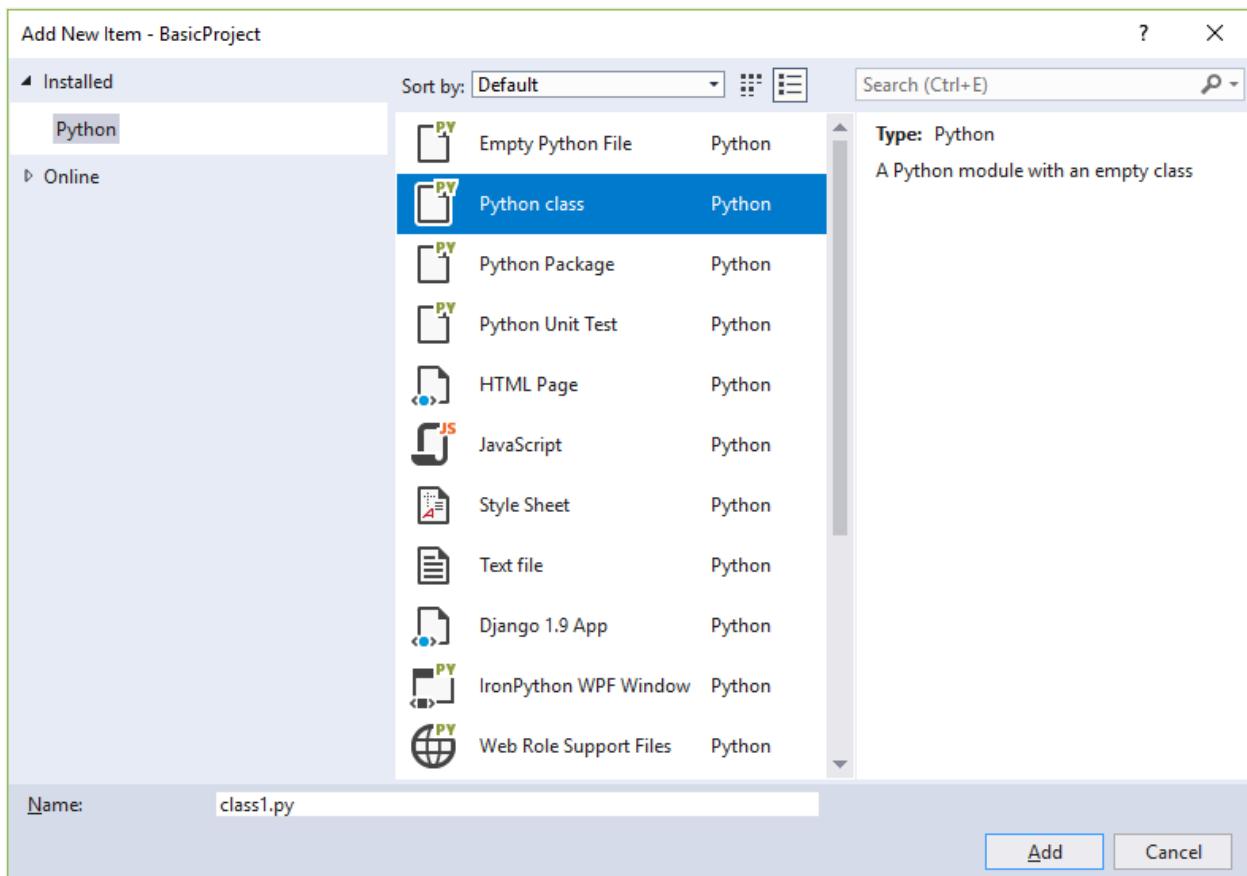
Nesta etapa, você aprenderá a:

- ✓ Usar modelos de item do Visual Studio para adicionar rapidamente novos arquivos de diferentes tipos com código clichê conveniente (etapa 3-1)
- ✓ Configurar o projeto em Django para fornecer arquivos estáticos (etapa 3-2)
- ✓ Adicionar mais páginas ao aplicativo (etapa 3-3)
- ✓ Usar a herança do modelo para criar um cabeçalho e uma barra de navegação usados nas páginas (etapa 3-4)

Etapa 3-1: Familiarizar-se com os modelos de item

À medida que você desenvolve um aplicativo Django, normalmente adiciona vários outros arquivos Python, HTML, CSS e JavaScript. Para cada tipo de arquivo (arquivos como `web.config`, que podem ser necessários para a implantação), o Visual Studio fornece [modelos de itens](#) convenientes para você começar.

Para exibir os modelos disponíveis, vá para **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta em que você deseja criar o item e, depois, selecione **Adicionar>Novo Item**.



Para usar um modelo, selecione o modelo desejado, especifique um nome para o arquivo e selecione **Adicionar**. A adição de um item dessa maneira adiciona automaticamente o arquivo ao projeto do Visual Studio e marca as alterações para controle do código-fonte.

Pergunta: Como o Visual Studio sabe quais modelos de item oferecer?

Resposta: O arquivo de projeto do Visual Studio (`.pyproj`) contém um identificador de tipo de projeto que o marca como um projeto do Python. O Visual Studio usa esse identificador de tipo para mostrar apenas os modelos de item que são adequados para o tipo de projeto. Dessa forma, o Visual Studio pode fornecer um conjunto avançado de modelos de item para muitos tipos de projeto sem solicitar que você os classifique toda vez.

Etapa 3-2: Fornecer arquivos estáticos do seu aplicativo

Em um aplicativo Web criado com Python (usando qualquer estrutura), seus arquivos em Python sempre são executados no servidor do host da Web. Os arquivos do Python também nunca são transmitidos para o computador de um usuário. Outros arquivos, no

entanto, como CSS e JavaScript, são usados exclusivamente pelo navegador. Portanto, o servidor host simplesmente os entrega como estão sempre que são solicitados. Esses arquivos são chamados de "estáticos", e o Django pode fornecê-los automaticamente sem que você precise escrever algum código.

Um projeto Django é configurado por padrão para fornecer arquivos estáticos da pasta `static` do aplicativo, graças a estas linhas no arquivo `settings.py` do projeto Django:

```
Python

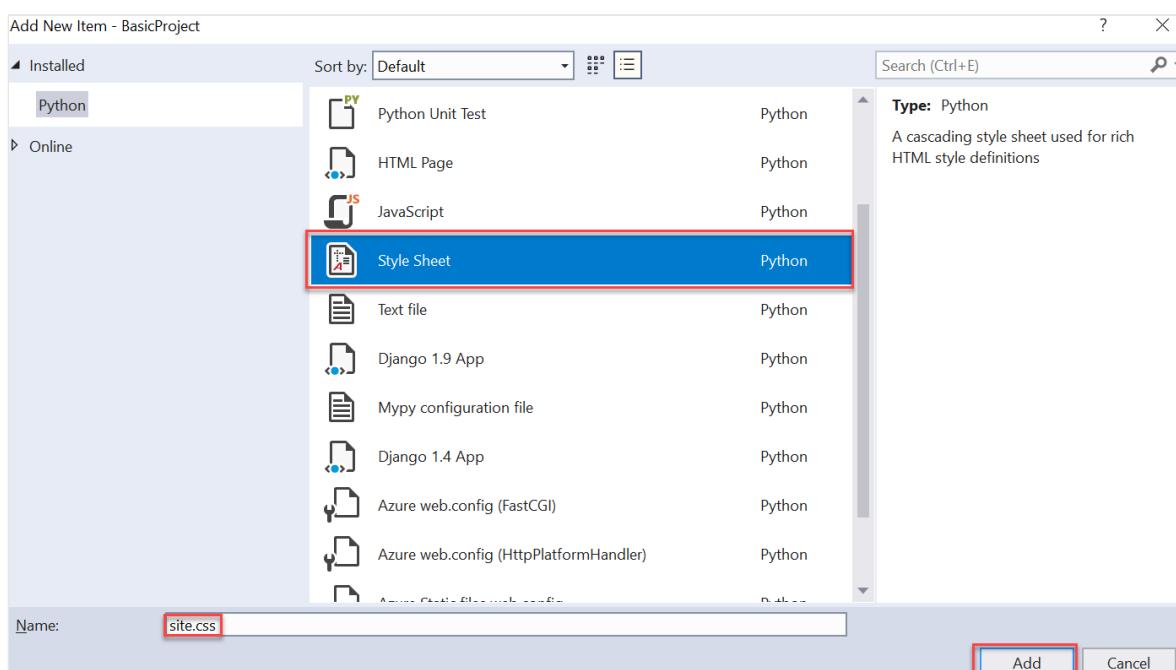
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC_URL = '/static/'

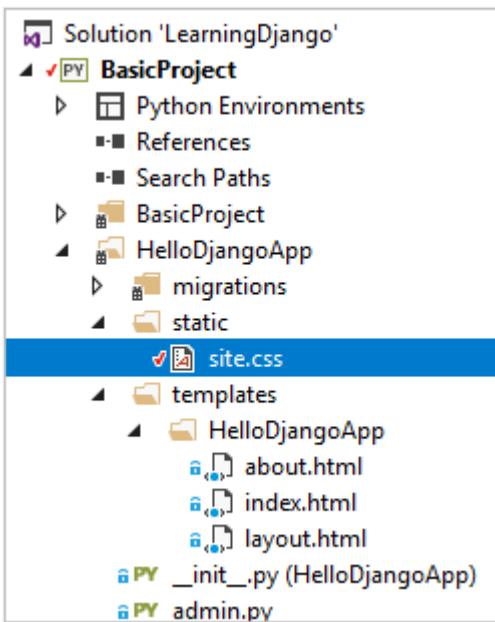
STATIC_ROOT = posixpath.join(*(BASE_DIR.split(os.path.sep) + ['static']))
```

Você pode organizar os arquivos dentro de `static` usando qualquer estrutura de pastas desejada e, em seguida, usar caminhos relativos dentro dessa pasta para referenciar os arquivos. Para demonstrar o processo, as seguintes etapas adicionam um arquivo CSS ao aplicativo e, em seguida, usam essa folha de estilos no modelo `index.html`:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta **HelloDjangoApp** do projeto do Visual Studio, selecione **Adicionar>Nova pasta** e nomeie a pasta `static`.
2. Clique com o botão direito do mouse na pasta `static` e selecione **Adicionar>Novo item**. Na caixa de diálogo exibida, selecione o modelo **Folha de estilos**, nomeie o arquivo `site.css` e selecione **Adicionar**.



O arquivo `site.css` é exibido no projeto e aberto no editor. A estrutura de pastas deve ser semelhante à imagem abaixo:



3. Substitua o conteúdo do arquivo `site.css` pelo seguinte código e salve o arquivo:

```
css

.message {
    font-weight: 600;
    color: blue;
}
```

4. Substitua o conteúdo do arquivo `templates/HelloDjangoApp/index.html` do aplicativo pelo código a seguir. O código substitui o elemento `` usado na etapa 2 por um `` que faz referência à classe de estilo `message`. O uso de uma classe de estilo oferece mais flexibilidade ao estilo do elemento. (Se você ainda não moveu o `index.html` para uma subpasta nos `modelos` ao usar o VS 2017 15.7 e anteriores, veja [Namespacing de modelo](#) na etapa 2-4.)

```
HTML

<html>
    <head>
        <title>{{ title }}</title>
        {% load static %} <!-- Instruct Django to load static files -->
        <link rel="stylesheet" type="text/css" href="{% static
'site.css' %}" />
    </head>
    <body>
        <span class="message">{{ message }}</span>{{ content }}
    </body>
</html>
```

5. Execute o projeto para observar os resultados. Quando estiver pronto, pare o servidor e confirme as alterações no controle do código-fonte se desejar (conforme explicado na [etapa 2](#)).

Pergunta: Qual é a finalidade da marcação `{% load static %}`?

Resposta: A linha `{% load static %}` é necessária antes da referência a arquivos estáticos em elementos como `<head>` e `<body>`. No exemplo mostrado nesta seção, "static files" se refere a um conjunto de marcações de modelo do Django personalizado, o que permite o uso da sintaxe `{% static %}` para se referir a arquivos estáticos. Sem `{% load static %}`, você verá uma exceção quando o aplicativo for executado.

Pergunta: Há convenções para organizar arquivos estáticos?

Resposta: Você pode adicionar outros arquivos CSS, JavaScript e HTML à sua pasta *static* como você quiser. Uma maneira comum de organizar arquivos estáticos é criar subpastas chamadas *fonts*, *scripts* e *content* (para folhas de estilo e outros arquivos). Em cada caso, lembre-se de incluir essas pastas no caminho relativo do arquivo nas referências `{% static %}`.

Pergunta: Posso concluir a mesma tarefa sem usar a marca `{% load static %}`?

Resposta: Sim, você pode.

```
HTML

<html>
  <head>
    <title>{{ title }}</title>
    <link rel="stylesheet" type="text/css" href="../../static/site.css"
  />
  </head>
  <body>
    <span class="message">{{ message }}</span>{{ content }}
  </body>
</html>
```

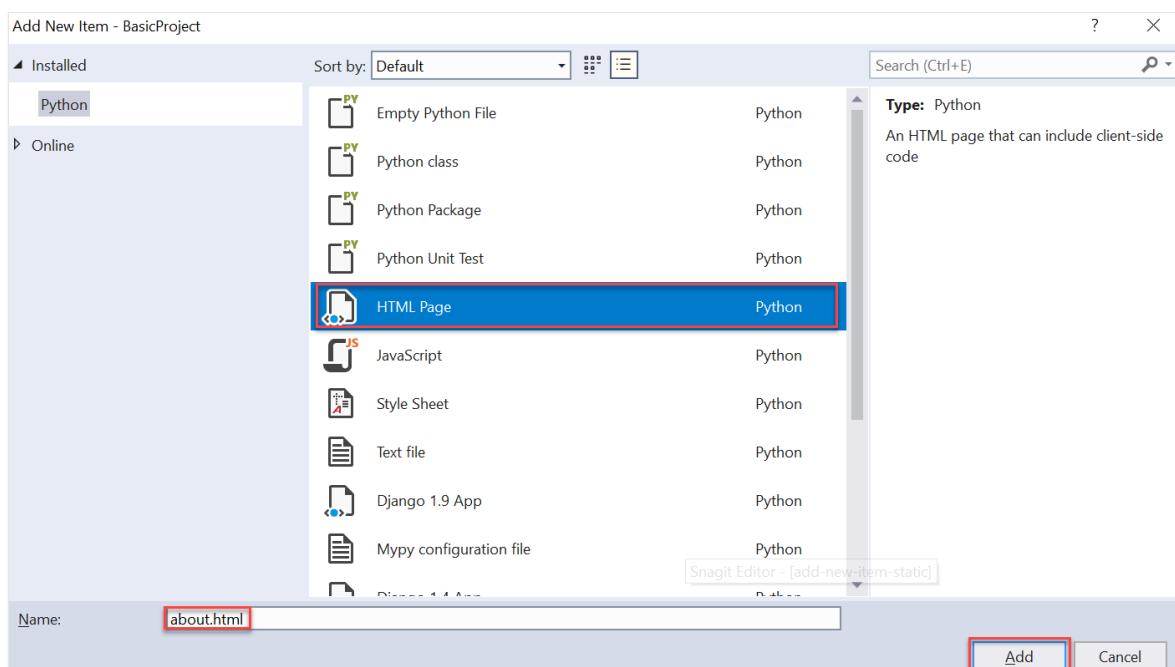
Etapa 3-3: Adicionar uma página ao aplicativo

A adição de outra página ao aplicativo fará com que:

- Adicione uma função em Python que defina o modo de exibição.
- Adicione um modelo para a marcação da página.
- Adicione o roteamento necessário ao arquivo `urls.py` do projeto do Django.

As etapas a seguir adicionam uma página "Sobre" ao projeto "HelloDjangoApp" e links para essa página na página inicial:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta `templates/HelloDjangoApp`. Selecione **Adicionar > Novo item** e selecione o modelo de item de página **HTML**. Nomeie o arquivo `about.html` e selecione **Adicionar**.



Dica

Se o comando **Novo Item** não aparecer no menu **Adicionar**, verifique se você interrompeu o servidor para que o Visual Studio saia do modo de depuração.

2. Substitua o conteúdo de `about.html` pela marcação abaixo (substitua o link explícito para a página inicial por uma barra de navegação simples na etapa 3-4):

```
HTML

<html>
  <head>
    <title>{{ title }}</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
      'site.css' %}" />
```

```
</head>
<body>
    <div><a href="home">Home</a></div>
    {{ content }}
</body>
</html>
```

3. Abra o arquivo *views.py* do aplicativo e adicione uma função chamada `about` que usa o modelo:

Python

```
def about(request):
    return render(
        request,
        "HelloDjangoApp/about.html",
        {
            'title' : "About HelloDjangoApp",
            'content' : "Example app page for Django."
        }
    )
```

4. Abra o arquivo *urls.py* do projeto do Django e adicione a seguinte linha à matriz

`urlPatterns`:

Python

```
re_path(r'^about$', HelloDjangoApp.views.about, name='about'),
```

5. Abra o arquivo *templates/HelloDjangoApp/index.html* e adicione a seguinte linha abaixo do elemento `<body>` para criar um link para a página About (substitua novamente esse link por uma barra de navegação na etapa 3-4):

HTML

```
<div><a href="about">About</a></div>
```

6. Salve todos os arquivos usando o comando de menu **Arquivo > Salvar Tudo** ou apenas pressione **Ctrl+Shift+S**. (Tecnicamente, essa etapa não é necessária, pois a execução do projeto no Visual Studio salva os arquivos automaticamente. No entanto, é um bom comando para se saber!)

7. Execute o projeto para observar os resultados e verificar a navegação entre as páginas. Quando terminar, feche o navegador.

Pergunta: Tentei usar "index" no link para a página inicial, mas não funcionou. Por quê?

Resposta: Embora a função de exibição no arquivo `views.py` seja nomeada `index`, os padrões de roteamento de URL no arquivo `urls.py` do projeto do Django não contêm uma expressão regular que corresponda à cadeia de caracteres "index". Para fazer a correspondência com a cadeia de caracteres, você precisa adicionar outra entrada para o padrão `^index$`.

Como mostrado na próxima seção, é melhor usar a marcação `{% url '<pattern_name>' %}` no modelo de página para se referir ao *nome* de um padrão. Nesse caso, o Django cria a URL adequada para você. Por exemplo, substitua `<div>Home</div>` em `about.html` por `<div>Home</div>`. O uso de 'index' funciona aqui porque o primeiro padrão de URL em `urls.py` é, na verdade, chamado de 'index' (em virtude do argumento `name='index'`). Você também pode usar "home" para se referir ao segundo padrão.

Etapa 3-4: Usar a herança do modelo para criar um cabeçalho e uma barra de navegação

Em vez de ter links de navegação explícitos em cada página, os aplicativos Web modernos usam um cabeçalho de identidade visual e uma barra de navegação. Uma barra de navegação fornece os links de página mais importantes, menus pop-up e assim por diante. Para garantir que o cabeçalho e a barra de navegação sejam os mesmos em todas as páginas, não repita o mesmo código em todos os modelos de página. Em vez disso, defina as partes comuns de todas as páginas em um único local.

O sistema de modelos do Django fornece dois meios para reutilização de elementos específicos em vários modelos: `includes` e `inheritance`.

- *Includes* corresponde a outros modelos de página que você insere em um local específico no modelo de referência usando a sintaxe `{% include <template_path>%}`. Se você quiser alterar o caminho dinamicamente no código, você pode também usar uma variável. Includes são usados no corpo de uma página para extrair o modelo compartilhado em um local específico na página.
- *Inheritance* usa o `{% extends <template_path>%}` no início de um modelo de página para especificar um modelo de base compartilhado no qual o modelo de referência se baseará. O elemento *Inheritance* costuma ser usado para definir um layout compartilhado, uma barra de navegação e outras estruturas para as páginas

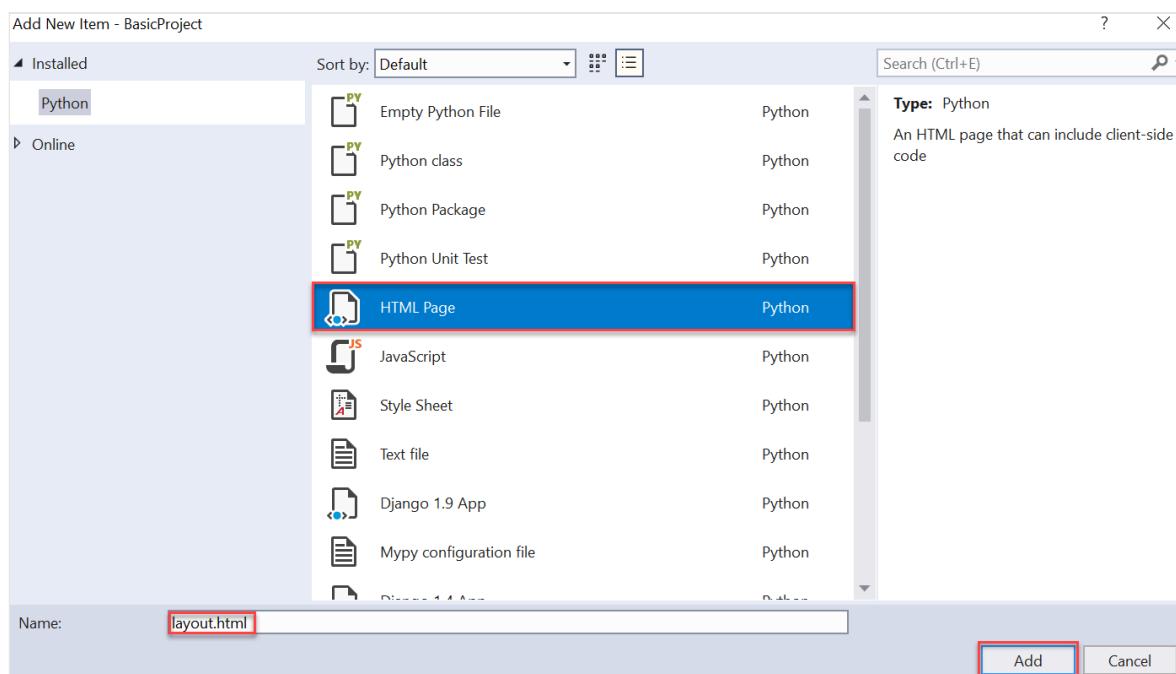
de um aplicativo, de modo que os modelos de referência possam apenas adicionar ou modificar áreas específicas do modelo de base chamadas *blocks*.

Em ambos os casos, `<template_path>` é relativo à pasta *templates* do aplicativo (`../` ou `./` também é permitido).

Um modelo de base delineia blocos usando as marcas `{% block <block_name> %}` e `{% endblock %}`. Se um modelo de referência usar marcações com o mesmo nome de bloco, o conteúdo do bloco substituirá o do modelo de base.

As etapas a seguir demonstram a herança:

1. Na pasta *templates/HelloDjangoApp* do aplicativo, crie um novo arquivo HTML. Clique com o botão direito do mouse na pasta **templates/HelloDjangoApp**, selecione **Adicionar>Novo item** e, em seguida, selecione o modelo de item **página HTML**. Nomeie o arquivo `layout.html` e selecione **Adicionar**.



2. Substitua o conteúdo do arquivo *layout.html* pela marcação abaixo. Veja que esse modelo contém um bloco chamado "conteúdo", que representa tudo o que as páginas de referência precisam substituir:

```
HTML

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>{{ title }}</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'site.css' %}" />
```

```

</head>

<body>
    <div class="navbar">
        <a href="/" class="navbar-brand">Hello Django</a>
        <a href="{% url 'home' %}" class="navbar-item">Home</a>
        <a href="{% url 'about' %}" class="navbar-item">About</a>
    </div>

    <div class="body-content">
        {% block content %}{% endblock %}
        <hr/>
        <footer>
            <p>&copy; 2018</p>
        </footer>
    </div>
</body>
</html>

```

3. Adicione os seguintes estilos ao arquivo *static/site.css* do aplicativo (este passo a passo não está tentando demonstrar o design responsivo; esses estilos servem apenas para gerar um resultado interessante):

css

```

.navbar {
    background-color: lightslategray;
    font-size: 1em;
    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
    color: white;
    padding: 8px 5px 8px 5px;
}

.navbar a {
    text-decoration: none;
    color: inherit;
}

.navbar-brand {
    font-size: 1.2em;
    font-weight: 600;
}

.navbar-item {
    font-variant: small-caps;
    margin-left: 30px;
}

.body-content {
    padding: 5px;
}

```

```
font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
}
```

4. Modifique o arquivo *templates/HelloDjangoApp/index.html* para se referir ao modelo base e torná-lo utilizável na página. Adicione a seguinte linha como linha 1 na página HTML (acima da marca html):

HTML

```
{% extends "HelloDjangoApp/layout.html" %}
```

5. Você pode ver que, usando a herança, esse modelo se torna simples de implementar dentro da marca body para substituir o bloco de conteúdo:

HTML

```
{% block content %}  
<span class="message">{{ message }}</span>{{ content }}  
{% endblock %}
```

6. Modifique o arquivo *templates/HelloDjangoApp/about.html* da mesma maneira para disponibilizar o modelo de layout. Adicione a mesma linha da etapa 1 na página HTML (acima da marca html):

HTML

```
{% extends "HelloDjangoApp/layout.html" %}
```

7. Em seguida, usando herança, implemente o modelo dentro da marca body para substituir o bloco de conteúdo:

HTML

```
{% block content %}  
{{ content }}  
{% endblock %}
```

8. Execute o servidor para observar os resultados. Quando terminar, feche o navegador.

Hello Django! on Thursday, 03 February, 2022 at 15:44:43

© 2018

9. Como você fez alterações significativas no aplicativo, é novamente um bom momento para [confirmar suas alterações no controle do código-fonte](#).

Próximas etapas

[Usar o modelo Projeto Web Django completo](#)

Aprofunde-se um pouco mais

- [Como gravar seu primeiro aplicativo do Django, parte 3 \(modos de exibição\)](#) ↗ (docs.djangoproject.com)
- Para conhecer mais recursos de modelos do Django, como o fluxo de controle, confira [A linguagem de modelos do Django](#) ↗ (docs.djangoproject.com)
- Para obter detalhes completos sobre como usar a marcação `{% url %}`, confira [url](#) ↗ dentro de [Referência de marcações de modelo internas e filtros para modelos do Django](#) ↗ (docs.djangoproject.com)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-django](#) ↗

Etapa 4: Usar o modelo Projeto Web do Django completo

Artigo • 16/03/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo](#)

Agora que explorou as noções básicas do Django no Visual Studio, você comprehende facilmente o aplicativo mais completo produzido pelo modelo "Projeto Web do Django".

Nesta etapa, você:

- ✓ Criar um aplicativo Web do Django mais completo usando o modelo "Projeto Web do Django" e examinar a estrutura do projeto (etapa 4-1)
- ✓ Entender os modos de exibição e os modelos de página criados pelo modelo de projeto, que consistem em três páginas que são herdadas a partir de um modelo de página de base e que emprega bibliotecas JavaScript estáticas como jQuery e Bootstrap (etapa 4-2)
- ✓ Entender o roteamento de URL fornecido pelo modelo (etapa 4-3)

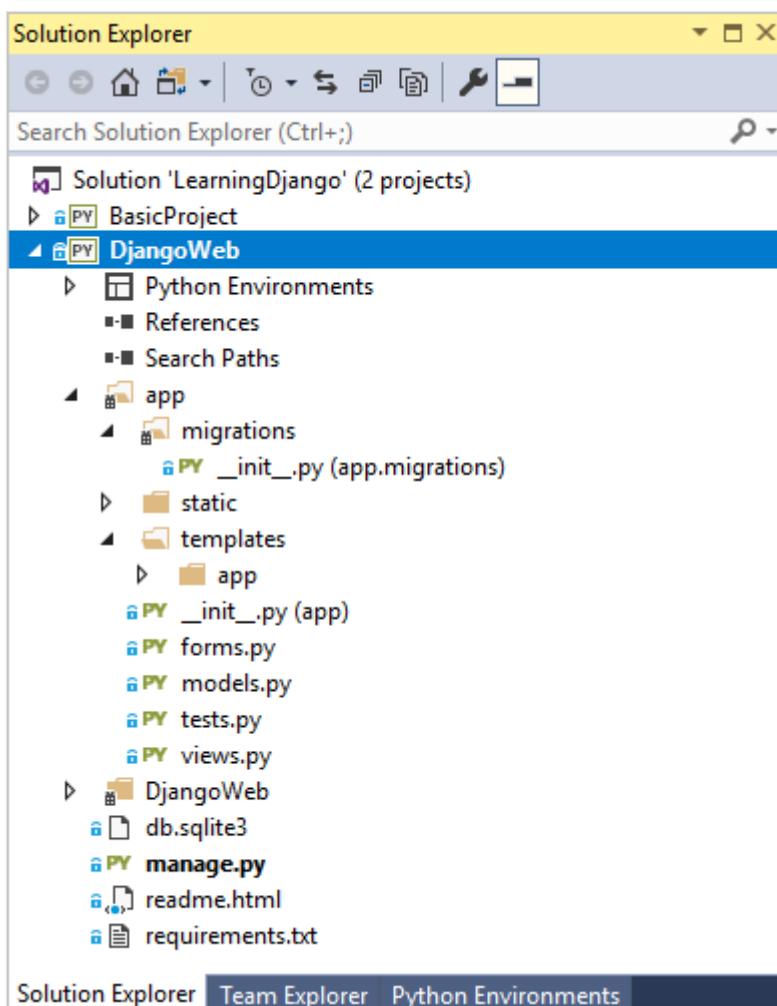
O modelo também fornece a autenticação básica, que é abordada na [Etapa 5](#).

Etapa 4-1: Criar um projeto com base no modelo

1. No Visual Studio, acesse o **Gerenciador de Soluções**, clique com o botão direito do mouse na solução **LearningDjango** criada anteriormente neste tutorial. Em seguida, selecione **Adicionar>Novo Projeto**. (Se você quiser usar uma nova solução, selecione **Arquivo>Novo>Projeto**).
2. Na caixa de diálogo **Novo Projeto**, pesquise e selecione o modelo **Projeto Web do Django**. Chame o projeto de "DjangoWeb" e selecione **Criar**.
3. Como o modelo inclui um arquivo *requirements.txt*, o Visual Studio solicita o local onde instalar as dependências. Quando solicitado, escolha a opção **Instalar em um ambiente virtual** e na caixa de diálogo **Adicionar Ambiente Virtual** selecione **Criar** para aceitar os padrões.

4. Quando o Visual Studio concluir a configuração do ambiente virtual, siga as instruções exibidas no arquivo *readme.html* para criar um superusuário do Django (ou seja, um administrador). Clique com o botão direito do mouse no projeto do Visual Studio e selecione o comando **Python>Criar Superusuário do Django** e, em seguida, siga os prompts. Registre seu nome de usuário e a senha que você usa ao exercitar os recursos de autenticação do aplicativo.

5. Defina o projeto **DjangoWeb** como o padrão para a solução do Visual Studio clicando com o botão direito do mouse no projeto em **Gerenciador de Soluções** e selecionando **Definir como Projeto de Inicialização**. O projeto de inicialização, mostrado em negrito, é executado quando você inicia o depurador.

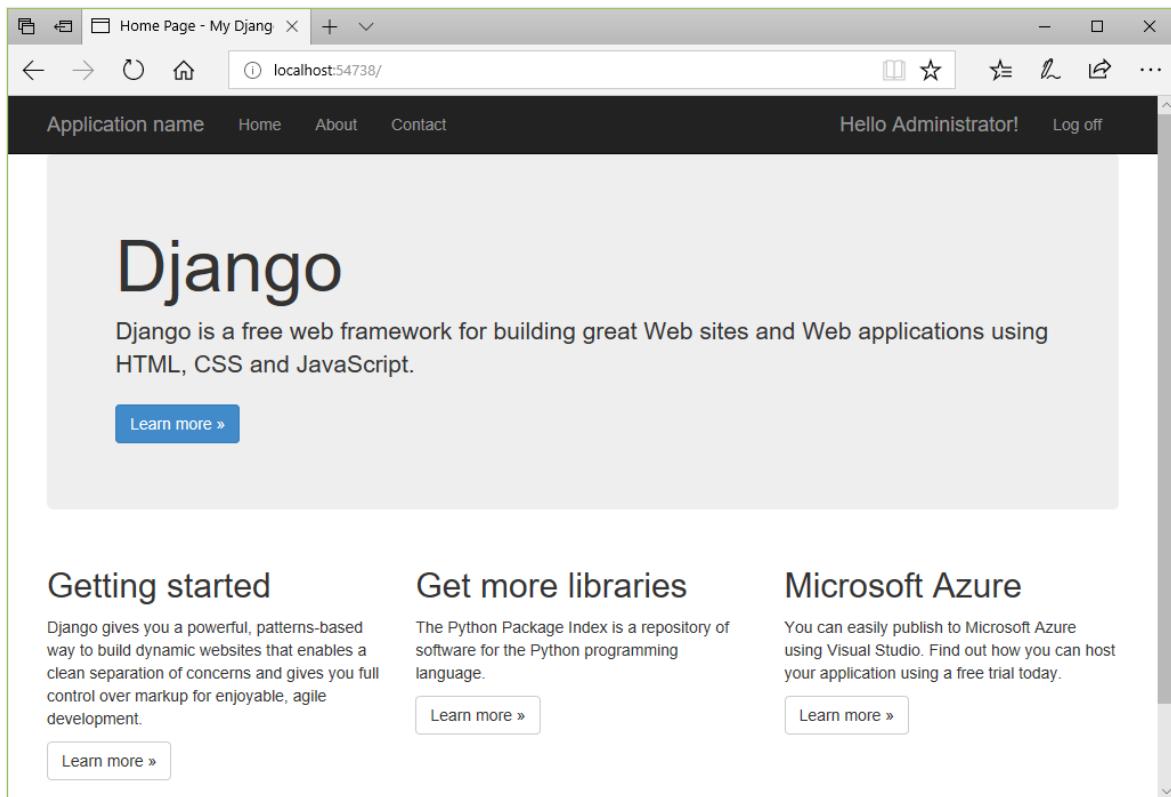


6. Selecione **Depurar>Iniciar Depuração (F5)** ou use o botão **Servidor Web** na barra de ferramentas para executar o servidor.

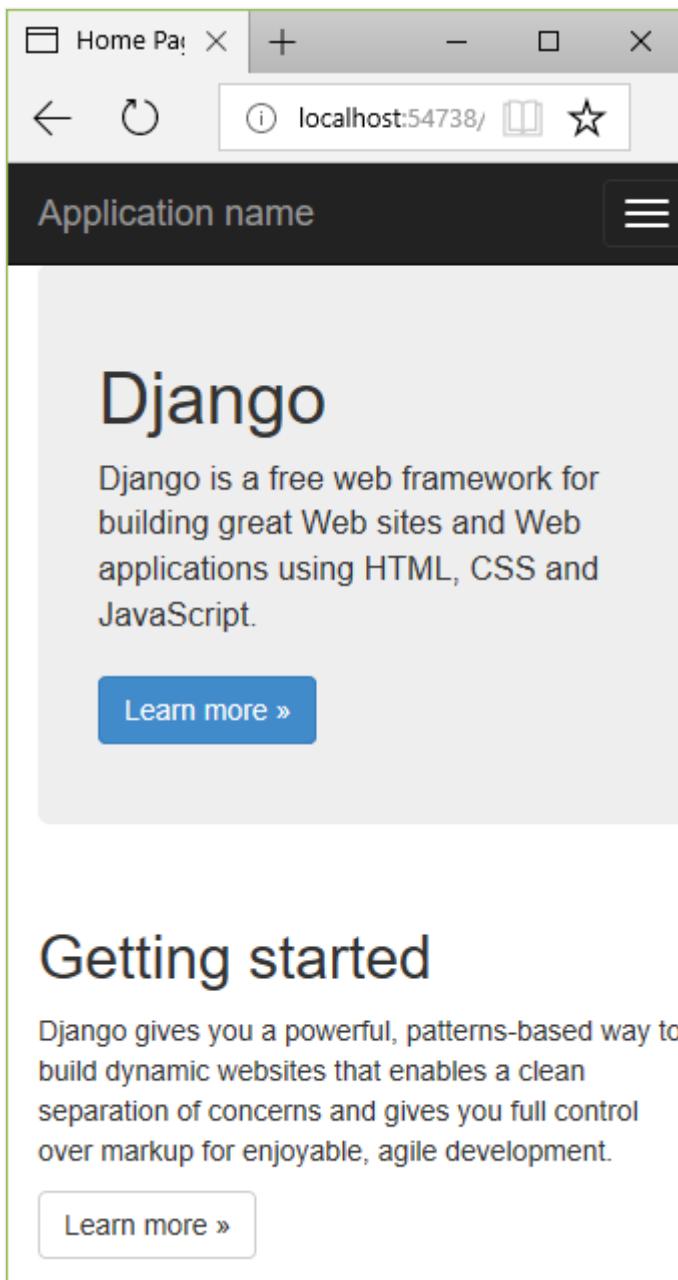


7. O aplicativo criado pelo modelo tem três páginas, Página Inicial, Sobre e Contato. Você pode navegar entre as páginas usando a barra de navegação. Levar um minuto ou dois para examinar as diferentes partes do aplicativo. Para autenticar

com o aplicativo por meio do comando **Login**, use as credenciais de superusuário criadas anteriormente.



8. O aplicativo criado pelo modelo "Projeto Web do Django" usa Bootstrap para layout responsivo que acomoda fatores forma móveis. Para ver essa capacidade de resposta, redimensione o navegador para uma exibição estreita para que o conteúdo seja renderizado verticalmente e a barra de navegação se transforme em um ícone de menu.



9. Você pode deixar o aplicativo em execução para as seções a seguir.

Caso deseje interromper o aplicativo e [confirmar as alterações no controle do código-fonte](#), abra a página **Alterações no Team Explorer**, clique com o botão direito do mouse na pasta do ambiente virtual (provavelmente **env**) e selecione **Ignorar estes itens locais**.

Examine o que o modelo cria

No nível mais amplo, o modelo "Projeto Web do Django" cria a seguinte estrutura:

- Arquivos na raiz do projeto:
 - *manage.py*: o utilitário administrativo do Django.
 - *db.sqlite3*: um banco de dados SQLite padrão.
 - *requirements.txt*: contém uma dependência do Django 1.x.

- *readme.html*: um arquivo exibido no Visual Studio após a criação do projeto.
Conforme observado na seção anterior, siga as instruções aqui para criar uma conta de superusuário (administrador) para o aplicativo.
- A pasta *app* contém todos os arquivos do aplicativo, incluindo exibições, modelos, testes, formulários, modelos e arquivos estáticos (veja a etapa 4-2). Normalmente, você renomeia esta pasta para usar um nome de aplicativo mais diferenciado.
- A pasta *DjangoWeb* (projeto do Django) contém os arquivos de projeto típicos do Django: *__init__.py*, *settings.py*, *urls.py* e *wsgi.py*. O arquivo *settings.py* já está configurado para o aplicativo e o arquivo de banco de dados usando o modelo de projeto. O arquivo *urls.py* também já está configurado com as rotas para todas as páginas do aplicativo, incluindo o formulário de entrada.

Pergunta: É possível compartilhar um ambiente virtual entre projetos do Visual Studio?

Resposta: Sim, no entanto, faça isso com a consciência de que projetos diferentes provavelmente usam pacotes diferentes ao longo do tempo. Portanto, um ambiente virtual compartilhado precisa conter todos os pacotes para todos os projetos que o usam.

No entanto, para usar um ambiente virtual existente, siga estas etapas:

1. Quando for solicitado a instalar dependências no Visual Studio, selecione a opção **Eu os instalarei por conta própria** opção.
2. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Ambientes do Python** e escolha **Adicionar Ambiente Virtual Existente**.
3. Navegue até e selecione a pasta que contém o ambiente virtual e selecione **OK**.

Etapa 4-2: Entender os modos de exibição e os modelos de página criados pelo modelo de projeto

Quando você executa o projeto, observe que o aplicativo contém três modos de exibição: Página inicial, Sobre e Contato. O código dessas exibições é encontrado no arquivo *views.py*. Cada função de exibição chama `django.shortcuts.render` com o caminho para um modelo e um objeto de dicionário simples. Por exemplo, a página Sobre é tratada pela função `about`:

```

def about(request):
    """Renders the about page."""
    assert isinstance(request, HttpRequest)
    return render(
        request,
        'app/about.html',
        {
            'title':'About',
            'message':'Your application description page.',
            'year':datetime.now().year,
        }
    )

```

Os modelos estão localizados na pasta `templates/app` do aplicativo (e, normalmente, você deseja renomear `app` como o nome do aplicativo real). O modelo base, `layout.html`, é o mais abrangente. O arquivo `layout.html` se refere a todos os arquivos estáticos necessários (JavaScript e CSS). O arquivo `layout.html` também define um bloco chamado "content" que outras páginas substituem e fornece outro bloco chamado "scripts". Os seguintes trechos anotados do arquivo `layout.html` mostram estas áreas específicas:

HTML

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />

    <!-- Define a viewport for Bootstrap's responsive rendering -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }} - My Django Application</title>

    {% load staticfiles %}
    <link rel="stylesheet" type="text/css" href="{% static
'app/content/bootstrap.min.css' %}" />
    <link rel="stylesheet" type="text/css" href="{% static
'app/content/site.css' %}" />
    <script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>
</head>
<body>
    <!-- Navbar omitted -->

    <div class="container body-content">

        <!-- "content" block that pages are expected to override -->
        {% block content %}{% endblock %}
        <hr/>
        <footer>
            <p>&copy; {{ year }} - My Django Application</p>
        </footer>
    </div>

```

```

<!-- Additional scripts; use the "scripts" block to add page-specific
scripts. -->
    <script src="{% static 'app/scripts/jquery-1.10.2.js' %}"></script>
    <script src="{% static 'app/scripts/bootstrap.js' %}"></script>
    <script src="{% static 'app/scripts/respond.js' %}"></script>
{% block scripts %}{% endblock %}

</body>
</html>

```

Cada um dos modelos de página individual, *about.html*, *contact.html* e *index.html*, estende o modelo base *layout.html*. O arquivo de modelo *about.html* é o mais simples e mostra as marcas `{% extends %}` e `{% block content %}`:

HTML

```

{% extends "app/layout.html" %}

{% block content %}

<h2>{{ title }}.</h2>
<h3>{{ message }}</h3>

<p>Use this area to provide additional information.</p>

{% endblock %}

```

Os arquivos de modelo *index.html* e *contact.html* usam a mesma estrutura e fornecem mais conteúdo no bloco "conteúdo".

Na pasta *templates/app*, também se encontra uma quarta página *login.html*, junto com *loginpartial.html*, que é inserida em *layout.html* com `{% include %}`. Esses arquivos de modelo são discutidos na etapa 5 na autenticação.

Pergunta: `{% block %}` e `{% endblock %}` podem ser recuados no modelo de página do Django?

Resposta: Sim. Os modelos de página do Django funcionam bem se você recua as marcas de bloco, talvez para alinhá-las nos elementos pai apropriados. Para exibir claramente onde as marcas de bloco são colocadas, os modelos de página do Visual Studio não recuam as marcas de bloco.

Etapa 4-3: Entender o roteamento de URL criado pelo modelo

O arquivo `urls.py` do projeto do Django criado pelo modelo "Projeto Web do Django" contém o seguinte código:

Python

```
from datetime import datetime
from django.urls import path
from django.contrib import admin
from django.contrib.auth.views import LoginView, LogoutView
from app import forms, views

urlpatterns = [
    path('', views.home, name='home'),
    path('contact/', views.contact, name='contact'),
    path('about/', views.about, name='about'),
    path('login/',
        LoginView.as_view(
            template_name='app/login.html',
            authentication_form=forms.BootstrapAuthenticationForm,
            extra_context={
                'title': 'Log in',
                'year' : datetime.now().year,
            }
        ),
        name='login'),
    path('logout/', LogoutView.as_view(next_page='/'), name='logout'),
    path('admin/', admin.site.urls),
]
```

Os três primeiros padrões de URL são mapeados diretamente para as exibições `home`, `contact` e `about` no arquivo `views.py` do aplicativo. Os padrões `^login/$` e `^logout$`, por outro lado, usam modos de exibição internos do Django em vez de exibições definidas pelo aplicativo. As chamadas para o método `url` também incluem dados adicionais para personalizar o modo de exibição. A Etapa 5 explora essas chamadas.

Pergunta: No projeto que criei, por que o padrão de URL "about" usa '^about' em vez de '^about\$', como mostrado aqui?

Resposta: A falta do '\$' à direita na expressão regular foi um erro simples em várias versões do modelo do projeto. O padrão de URL funciona perfeitamente para uma página chamada "sobre". No entanto, sem o '\$' à direita, o padrão de URL também corresponde a URLs como "about=django," "about09876," "aboutoflaughter," e assim

por diante. O '\$' à direita é mostrado aqui para criar um padrão de URL que corresponde *somente* a "about".

Próximas etapas

Autenticar usuários no Django

Aprofunde-se um pouco mais

- Como gravar seu primeiro aplicativo do Django, parte 4 – Modos de exibição genéricos e formulários ↗ (docs.djangoproject.com)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-django](#) ↗

Etapa 5: Autenticar usuários no Django

Artigo • 10/04/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Usar o modelo Projeto Web completo do Django](#)

O modelo "Projeto Web do Django" inclui um fluxo de autenticação básico, já que a autenticação é uma necessidade comum para aplicativo Web. Ao usar um dos modelos de projeto do Django, o Visual Studio inclui todos os módulos necessários para a autenticação no arquivo `settings.py` do projeto do Django.

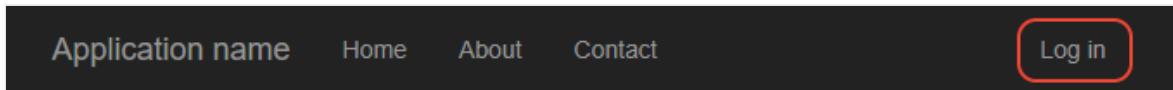
Nesta etapa, você aprenderá:

- ✓ Como usar o fluxo de autenticação fornecido em modelos do Visual Studio (etapa 5 – 1)

Etapa 5-1: Usar o fluxo de autenticação

As etapas a seguir acionam o fluxo de autenticação e descrevem as partes do projeto:

1. Se você ainda não seguiu as instruções do arquivo `readme.html` na raiz do projeto para criar uma conta de superusuário (administrador), faça isso agora.
2. Execute o aplicativo no Visual Studio usando **Depurar>Iniciar Depuração (F5)**. Quando o aplicativo aparecer no navegador, observe que a opção **Login** aparecerá no canto superior direito da barra de navegação.



3. Abra `templates/app/layout.html` e observe que o elemento `<div class="navbar" ...>` contém a marcação `{% include app/loginpartial.html %}`. A marcação `{% include %}` instrui o sistema de modelos do Django a efetuar pull do conteúdo do arquivo incluído neste momento no modelo que o contém.
4. Abra `templates/app/loginpartial.html` e observe como ele usa a marcação condicional `{% if user.is_authenticated %}` juntamente com uma marcação `{% else %}` para renderizar diferentes elementos da interface do usuário, dependendo se o usuário realizou a autenticação:

HTML

```

{% if user.is_authenticated %}
<form id="logoutForm" action="/logout" method="post" class="navbar-right">
    {% csrf_token %}
    <ul class="nav navbar-nav navbar-right">
        <li><span class="navbar-brand">Hello {{ user.username }}!
    </span></li>
        <li><a href="javascript:document.getElementById('logoutForm').submit()">Log off</a></li>
    </ul>
</form>

{% else %}

<ul class="nav navbar-nav navbar-right">
    <li><a href="{% url 'login' %}">Log in</a></li>
</ul>

{% endif %}

```

5. Como nenhum usuário é autenticado quando você inicia o aplicativo pela primeira vez, esse código de modelo renderiza apenas o link "Fazer logon" para o caminho "login" relativo. Conforme especificado em `urls.py` e exibido na seção anterior, essa rota é mapeada para o modo de exibição `django.contrib.auth.views.login` que recebe os seguintes dados:

```

Python

{
    'template_name': 'app/login.html',
    'authentication_form': app.forms.BootstrapAuthenticationForm,
    'extra_context':
    {
        'title': 'Log in',
        'year': datetime.now().year,
    }
}

```

Aqui, `template_name` identifica o modelo da página de logon, nesse caso, `templates/app/login.html`. A propriedade `extra_context` é adicionada aos dados de contexto padrão fornecidos para o modelo. Por fim, `authentication_form` especifica uma classe de formulário a ser usada com o logon; no modelo, ele é exibido como o objeto `form`. O valor padrão é `AuthenticationForm` (de `django.contrib.auth.views`). Em vez disso, o modelo de projeto do Visual Studio usa o formulário definido no arquivo `forms.py` do aplicativo:

Python

```
from django import forms
from django.contrib.auth.forms import AuthenticationForm
from django.utils.translation import ugettext_lazy as _

class BootstrapAuthenticationForm(AuthenticationForm):
    """Authentication form which uses bootstrap CSS."""
    username = forms.CharField(max_length=254,
                               widget=forms.TextInput({
                                   'class': 'form-control',
                                   'placeholder': 'User name'}))
    password = forms.CharField(label=_("Password"),
                               widget=forms.PasswordInput({
                                   'class': 'form-control',
                                   'placeholder': 'Password'}))
```

Como você pode ver, a classe de formulário deriva de `AuthenticationForm` e, especificamente, substitui os campos de nome de usuário e senha para adicionar texto de espaço reservado. O modelo do Visual Studio inclui esse código explícito considerando que você possa querer personalizar o formulário, por exemplo, adicionando uma validação de força de senha.

6. Quando você navega para a página de logon, o aplicativo renderiza o modelo `login.html`. As variáveis `{{ form.username }}` e `{{ form.password }}` renderizam os formulários `CharField` de `BootstrapAuthenticationForm`. Há também uma seção interna para mostrar erros de validação e um elemento predefinido para efetuar logons em redes sociais, se você optar por adicionar esses serviços.

HTML

```
{% extends "app/layout.html" %}

{% block content %}

<h2>{{ title }}</h2>
<div class="row">
    <div class="col-md-8">
        <section id="loginForm">
            <form action"." method="post" class="form-horizontal">
                {% csrf_token %}
                <h4>Use a local account to log in.</h4>
                <hr />
                <div class="form-group">
                    <label for="id_username" class="col-md-2 control-label">User name</label>
                    <div class="col-md-10">
                        {{ form.username }}
                    </div>
```

```

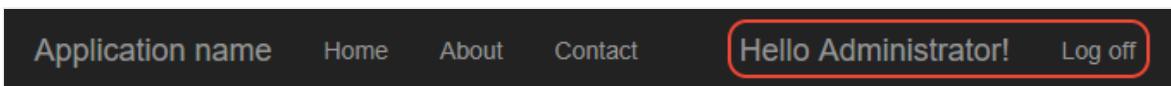
        </div>
        <div class="form-group">
            <label for="id_password" class="col-md-2 control-
label">Password</label>
            <div class="col-md-10">
                {{ form.password }}
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="hidden" name="next" value="/" />
                <input type="submit" value="Log in" class="btn
btn-default" />
            </div>
        </div>
    {% if form.errors %}
        <p class="validation-summary-errors">Please enter a
correct user name and password.</p>
    {% endif %}
    </form>
</section>
</div>
<div class="col-md-4">
    <section id="socialLoginForm"></section>
</div>
</div>

{% endblock %}

```

7. Quando você envia o formulário, o Django tenta autenticar suas credenciais (como as credenciais do superusuário). Se a autenticação falhar, você permanecerá na página atual, porém `form.errors` será definido como true. Se a autenticação for bem-sucedida, o Django navegará para a URL relativa no campo "avançar", `<input type="hidden" name="next" value="/" />`, que, nesse caso, é a página inicial (/).

8. Agora, quando a home page for novamente renderizada, a propriedade `user.is_authenticated` será verdadeira quando o modelo `loginpartial.html` for renderizado. Como resultado, você verá uma mensagem Olá, (nome de usuário) e Fazer logoff. Você pode usar `user.is_authenticated` em outras partes do aplicativo para verificar a autenticação.



9. Você precisa recuperar as permissões específicas do usuário do banco de dados para verificar se o usuário autenticado está autorizado a acessar recursos específicos. Para obter mais informações, confira [Using the Django authentication system](#) (Usando o sistema de autenticação do Django) (documentos do Django).

10. O superusuário ou o administrador, em particular, está autorizado a acessar as interfaces de administrador internas do Django usando as URLs relativas "/admin/" e "/admin/doc/". Para habilitar essas interfaces, siga as etapas abaixo:

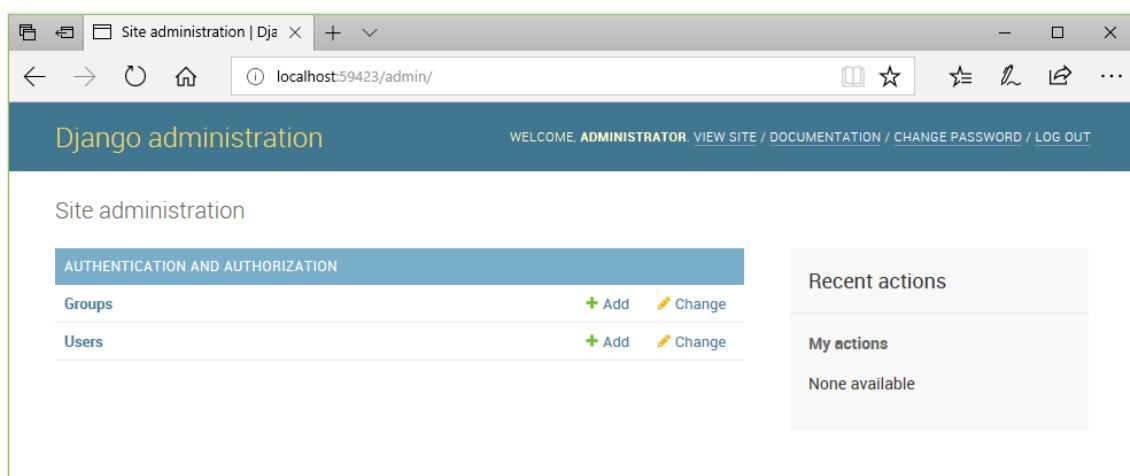
- a. Instale o pacote Python docutils em seu ambiente. Uma ótima maneira de instalar é adicionar "docutils" ao arquivo *requirements.txt*. Em seguida, no **Gerenciador de Soluções**, expandir o projeto, expandir o nó **Ambientes do Python** e, em seguida, clicar com o botão direito do mouse no ambiente que você está usando e selecionar **Instalar do requirements.txt**.
- b. Abra o arquivo *urls.py* do projeto do Django e adicione o seguinte:

```
Python

from django.conf.urls import include
from django.contrib import admin
admin.autodiscover()

urlpatterns = [
    path('admin/doc/', include('django.contrib.admindocs.urls'))
]
```

- c. No arquivo *settings.py* do projeto do Django, navegue até a coleção **INSTALLED_APPS** e adicione '`'django.contrib.admindocs'`'.
- d. Quando você reiniciar o aplicativo, navegue até "/admin/" e "/admin/doc/" e execute tarefas, como criar mais contas de usuário.



11. A parte final do fluxo de autenticação é fazer logoff. Como você pode ver em *loginpartial.html*, o link **Fazer logoff** apenas faz um POST para a URL relativa "/login", que é manipulada pela exibição interna `django.contrib.auth.views.logout`. Essa exibição não exibe nenhuma interface do usuário e apenas navega para a home page (conforme mostrado em *urls.py* para o padrão "`^logout$`"). Se você quiser exibir uma página de logoff, primeiro altere o

padrão da URL conforme mostrado a seguir para adicionar uma propriedade "template_name" e remover a propriedade "next_page":

```
Python

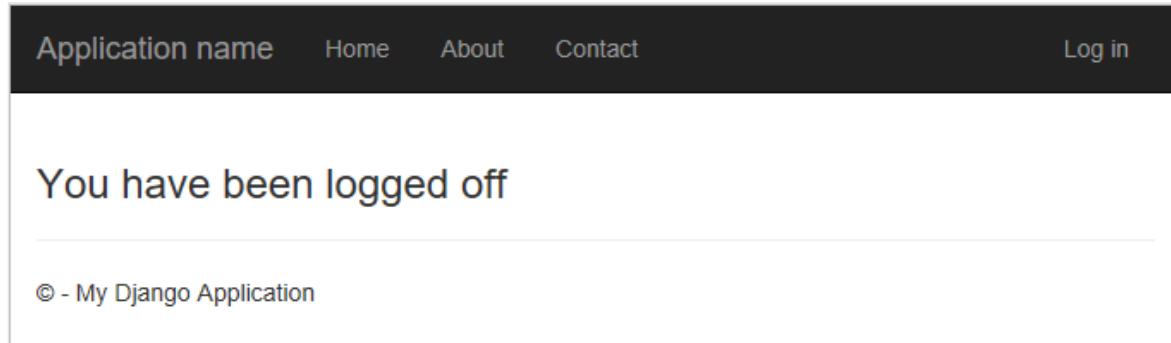
path('logout/',
      django.contrib.auth.views.logout,
      {
          'template_name': 'app/loggedoff.html',
          # 'next_page': '/',
      },
      name='logout')
```

Em seguida, crie *templates/app/loggedoff.html* com o seguinte conteúdo (mínimo):

```
HTML

{% extends "app/layout.html" %}
{% block content %}
<h3>You have been logged off</h3>
{% endblock %}
```

O resultado se parece com o seguinte:



12. Quando terminar, interrompa o servidor e, mais uma vez, confirme suas alterações no controle do código-fonte.

Pergunta: Qual é a finalidade da marcação `{% csrf_token %}` exibida nos elementos `<form>`?

Resposta: A marcação `{% csrf_token %}` inclui a [proteção interna contra solicitações intersetores forjadas \(csrf\)](#) do Django (documentos do Django). Normalmente, essa marca é adicionada a qualquer elemento que envolva métodos de solicitação POST, PUT ou DELETE, como um formulário. Em seguida, a função de renderização de modelo (`render`) insere a proteção necessária.

Próximas etapas

① Observação

Se você tiver confirmado sua solução do Visual Studio para controle de origem ao longo deste tutorial, agora será um bom momento para fazer outra confirmação. A solução deve corresponder ao código-fonte do tutorial no GitHub:

[Microsoft/python-sample-vs-learning-django ↗](#).

Agora você explorou a totalidade dos modelos de "Projeto em branco da Web do Django" e "Pesquisas de Projeto Web do Django" no Visual Studio. Você também aprendeu todas as noções básicas do Django, como usar exibições e modelos. Além disso, explorou o roteamento, a autenticação e os modelos de banco de dados usados. Agora você deverá ser capaz de criar um aplicativo Web por sua conta com os modos de exibição e os modelos de que precisar.

A execução de um aplicativo Web no computador de desenvolvimento é apenas uma etapa para disponibilizar o aplicativo para seus clientes. As próximas etapas podem incluir as seguintes tarefas:

- Personalizar a página 404 criando um modelo chamado *templates/404.html*. Quando presente, o Django usa esse modelo, em vez de seu padrão. Para saber mais, veja [Modos de exibição de erro ↗](#) na documentação do Django.
- Escreva testes de unidade em *tests.py*; os modelos de projeto do Visual Studio fornecem pontos iniciais para eles e mais informações podem ser encontradas em [Escrevendo seu primeiro aplicativo do Django, parte 5 – teste ↗](#) e [Testando no Django ↗](#) na documentação do Django.
- Altere o aplicativo de SQLite para um repositório de dados de nível de produção como PostgreSQL, MySQL e SQL Server (que pode ser hospedado no Azure). Conforme descrito em [Quando usar o SQLite ↗](#) (sqlite.org), o SQLite funciona bem para sites de tráfego baixo a médio com menos de 100 mil acessos por dia. No entanto, o SQLite não é recomendado para volumes mais altos. O SQLite também é limitado a um único computador, de modo que ele não pode ser usado em qualquer cenário de vários servidores, como balanceamento de carga e replicação geográfica. Para obter informações sobre o suporte do Django para outros bancos de dados, veja [Instalação do banco de dados ↗](#). Você também pode usar o [SDK do Azure para Python](#) para trabalhar com serviços de armazenamento do Azure como tabelas e blobs.

- Configure um pipeline de integração contínua/implantação contínua em um serviço como o Azure DevOps. Além de trabalhar com o controle do código-fonte (por meio do Azure Repos, do GitHub ou em outro local), você pode configurar um projeto do Azure DevOps para executar automaticamente seus testes de unidade como um pré-requisito para a versão. Você também pode configurar o pipeline para implantar em um servidor de preparo para mais testes antes de implantar na produção. O Azure DevOps, além disso, integra-se às soluções de monitoramento, como o App Insights e fecha o ciclo de inteiro com ferramentas ágeis de planejamento. Para saber mais, confira [Criar um pipeline de CI/CD para Python com o projeto do Azure DevOps](#) e também a [documentação geral do Azure DevOps](#).

Aprofunde-se um pouco mais

- [Autenticação de usuário no Django ↗](#) (docs.djangoproject.com)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-django ↗](#)

Tutorial: Introdução à estrutura da Web do Flask no Visual Studio

Artigo • 08/09/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

O [Flask](#) é uma estrutura leve do Python para aplicativos Web que fornece as noções básicas de roteamento de URL e renderização de página.

O Flask é denominado uma "microestrutura", porque não oferece diretamente funcionalidades como validação de formulário, abstração de banco de dados, autenticação e assim por diante. Essas funcionalidades são fornecidas por pacotes especiais do Python chamados *extensões* do Flask. As extensões integram-se perfeitamente ao Flask para serem exibidas como se fizessem parte do próprio Flask. Por exemplo, o Flask não oferece um mecanismo de modelo de página. A modelagem é fornecida por extensões como Jinja e Jade, conforme demonstrado neste tutorial.

Neste tutorial, você aprenderá como:

- Crie um projeto básico do Flask em um repositório GIT usando o modelo "Projeto Web em Branco do Flask" (etapa 1).
- Crie um aplicativo do Flask com uma página e renderize essa página usando um modelo (etapa 2).
- Forneça arquivos estáticos, adicione páginas e use a herança do modelo (etapa 3).
- Use o modelo Projeto Web do Flask para criar um aplicativo com várias páginas e design responsivo (etapa 4).

No decorrer dessas etapas, você criará uma única solução do Visual Studio que contém dois projetos separados. Você criará o projeto usando diferentes modelos de projeto do Flask incluídos com o Visual Studio. Se você mantiver os projetos na mesma solução, poderá facilmente alternar entre diferentes arquivos para comparação.

Observação

Este tutorial é diferente do [Início Rápido do Flask](#) no qual você saberá mais sobre o Flask e como usar os diferentes modelos de projeto do Flask que fornecem um ponto de início mais amplo para seus próprios projetos. Por exemplo, os modelos de projeto instalam automaticamente o pacote do Flask ao criar um projeto, em vez de precisar instalar o pacote manualmente, conforme mostrado no Início Rápido.

Pré-requisitos

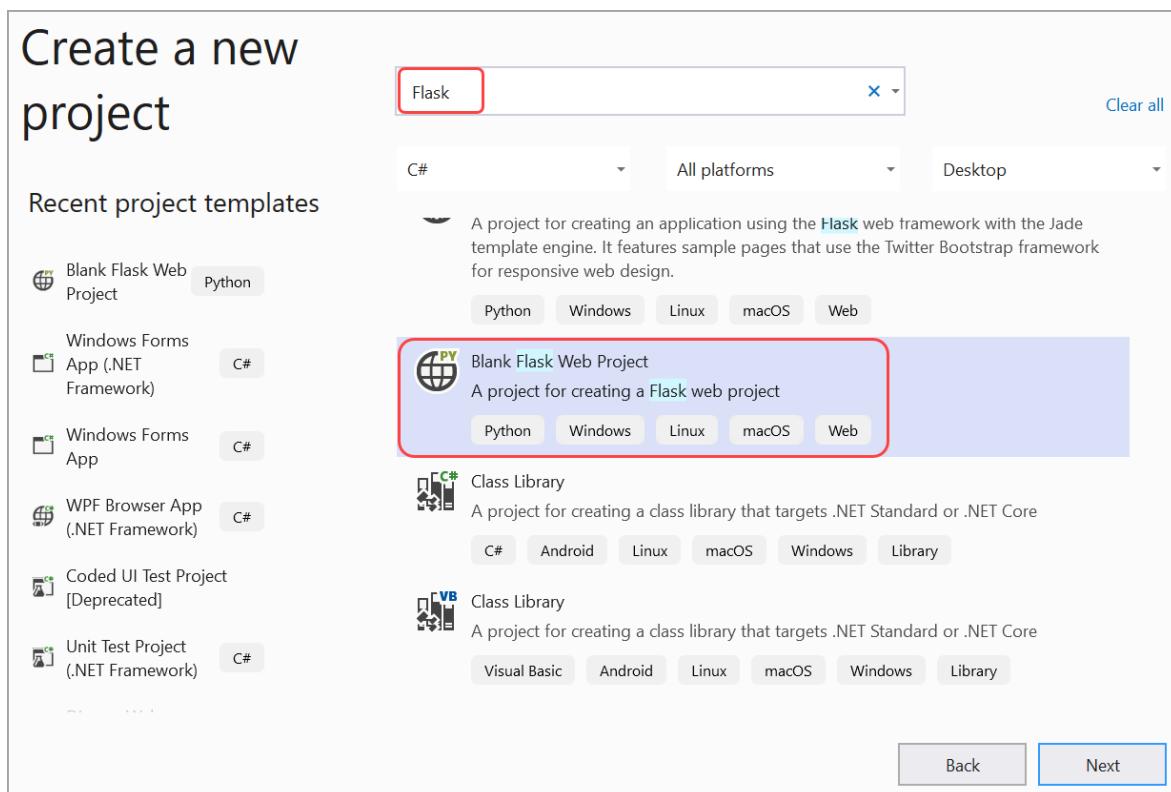
- Visual Studio 2022 no Windows com as seguintes opções:
 - A carga de trabalho **desenvolvimento do Python** (guia **Carga de Trabalho** no instalador). Para obter instruções, confira [Instalar o suporte do Python no Visual Studio](#).
 - **Git para Windows** na guia **Componentes individuais** em **Ferramentas de código**.

Os modelos de projeto do Flask são incluídos com todas as versões anteriores das Ferramentas Python para Visual Studio, embora os detalhes possam ser diferentes do que foi discutido neste tutorial.

No momento, não há suporte para o desenvolvimento do Python no Visual Studio para Mac. No Mac e no Linux, use o tutorial [Extensão Python no Visual Studio Code](#).

Etapa 1-1: Criar uma solução e um projeto do Visual Studio

1. No Visual Studio, selecione **Arquivo > Novo > Projeto** e pesquise por "Flask". Em seguida, selecione o modelo em **Projeto Web em branco do Flask** e Avançar.



2. Configure seu novo projeto inserindo as seguintes informações e selecione **Criar**:

- **Nome:** defina o nome do projeto do Visual Studio como **BasicProject**. Esse nome também é usado para o projeto do Flask.
- **Local:** especifique um local no qual criar o projeto e a solução do Visual Studio.
- **Nome da solução:** definido como **LearningFlask**, apropriado para a solução como um contêiner para vários projetos neste tutorial.

Etapa 1-2: Examinar os controles do Git e publicar em um repositório remoto

Nesta etapa, familiarize-se com os controles do Git do Visual Studio e a janela do **Team Explorer** onde você trabalha com o controle do código-fonte.

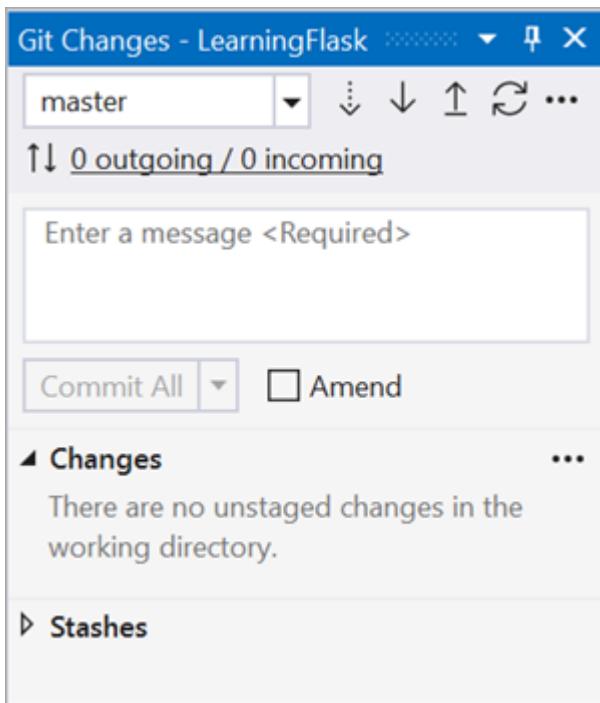
1. Para confirmar o projeto no controle do código-fonte local, selecione o comando **Adicionar ao Controle do Código-Fonte** no canto inferior da janela principal do Visual Studio e selecione a opção Git. Essa ação levará você para a janela Criar repositório Git, na qual poderá criar e enviar por push um novo repositório.



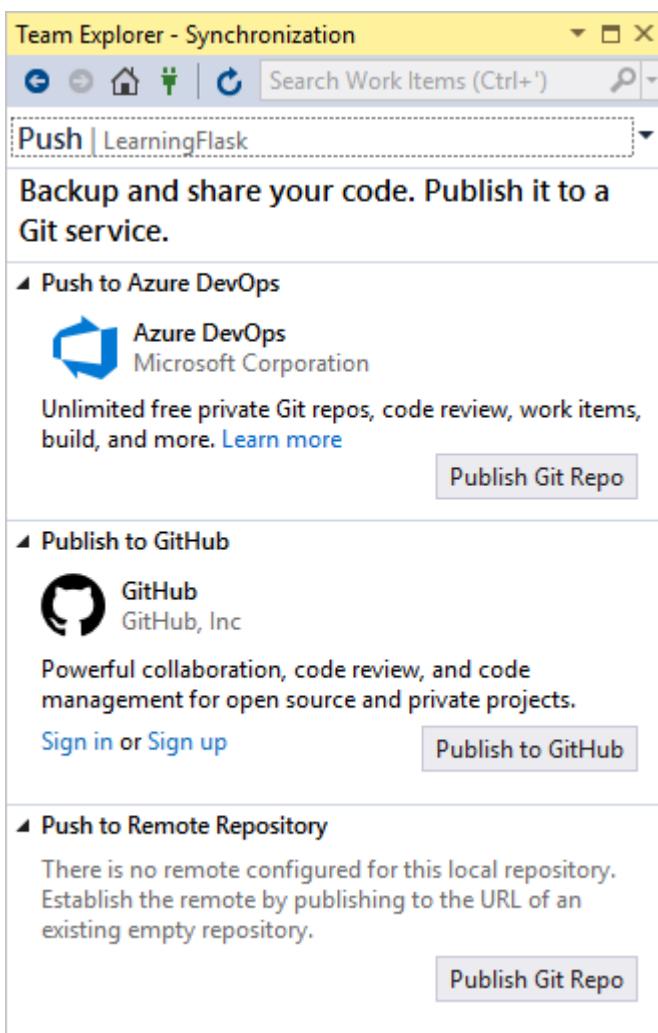
2. Depois de criar um repositório, um conjunto de novos controles do Git aparecerá na parte inferior. Da esquerda para direita, esses controles mostram commits não enviados, alterações não confirmadas, o branch atual e o nome do repositório:



3. Marque o botão de alterações do Git, e o Visual Studio abrirá a janela do **Team Explorer** na página **Alterações do Git**. Como o projeto recém-criado já está automaticamente confirmado no controle do código-fonte, você não verá as alterações pendentes.

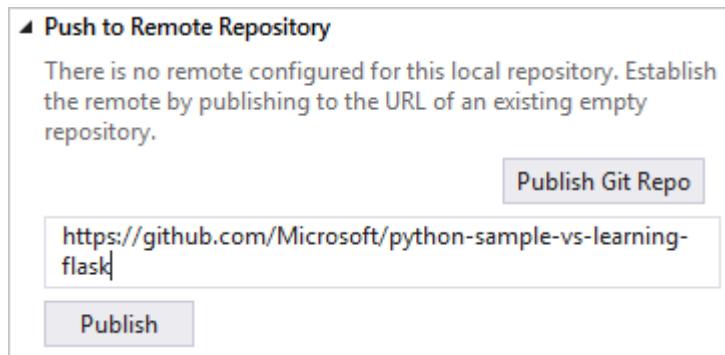


4. Na barra de status do Visual Studio, marque o botão de confirmação não enviada (a seta para cima com um 2) para abrir a página **Sincronização** no **Team Explorer**. Como você tem apenas um repositório local, a página oferece opções simples para publicar o repositório em diferentes repositórios remotos.



Você pode selecionar o serviço que desejar para seus próprios projetos. Este tutorial mostra o uso do GitHub, em que o código de exemplo concluído do tutorial é mantido no repositório [Microsoft/python-sample-vs-learning-django](https://github.com/Microsoft/python-sample-vs-learning-django).

5. Ao selecionar qualquer um dos controles **Publicar**, o **Team Explorer** solicitará mais informações. Por exemplo, ao publicar o exemplo para este tutorial, primeiro foi necessário criar o próprio repositório. Nesse caso, a opção **Enviar por Push para Re却itório Remoto** foi usada com a URL do repositório.



Se você não tiver um repositório, as opções **Publicar no GitHub** e **Enviar por Push para o Azure DevOps** permitirão criar um repositório diretamente no Visual Studio.

6. Ao trabalhar com este tutorial, adquira o hábito de usar periodicamente os controles no Visual Studio para confirmar e enviar alterações por push. Este tutorial envia lembretes para você nos pontos apropriados.

Dica

Para navegar rapidamente no **Team Explorer**, selecione o cabeçalho (que indica **Alterações** ou **Efetuar Push** nas imagens acima) para ver um menu pop-up das páginas disponíveis.

Pergunta: Quais são algumas vantagens de usar o controle do código-fonte a partir do início de um projeto?

Reposta: o uso do controle do código-fonte desde o início, especialmente se você também usar um repositório remoto, fornece um backup regular do projeto em um local externo. Em vez de manter um projeto apenas em um sistema de arquivos local, o controle do código-fonte também oferece um histórico de alterações completo e facilita a reversão de um único arquivo ou todo o projeto para um estado anterior. O histórico de alterações ajuda a determinar a causa das regressões (falhas de teste). O controle do

código-fonte é essencial se várias pessoas estiverem trabalhando em um projeto, pois ele gerencia substituições e fornece a resolução de conflitos. O controle do código-fonte é basicamente uma forma de automação, preparando para automatizar o gerenciamento de compilações, testes e versões. Esse recurso representa o primeiro passo no uso de DevOps em um projeto e, como os obstáculos para entrar são muito baixos, realmente não há motivos para não usar o controle do código-fonte desde o início.

Para uma discussão mais aprofundada sobre o controle do código-fonte usado como automação, confira [A origem da verdade: a função dos repositórios no DevOps](#), um artigo da MSDN Magazine destinado a aplicativos móveis, mas que também se aplica a aplicativos Web.

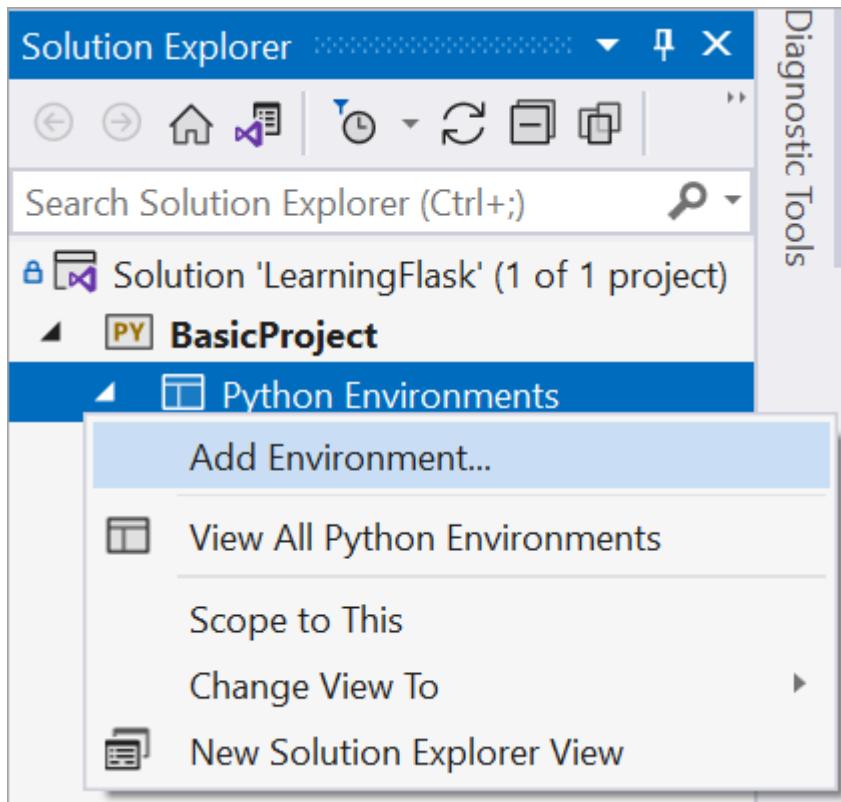
Pergunta: É possível evitar que o Visual Studio confirme automaticamente um novo projeto?

Resposta: Sim. Para desabilitar a confirmação automática, vá para a página **Configurações** no **Team Explorer**, escolha **Git>Configurações Globais**, desmarque o opção rotulada como **Confirmar alterações após mesclagem por padrão** e, em seguida, escolha **Atualizar**.

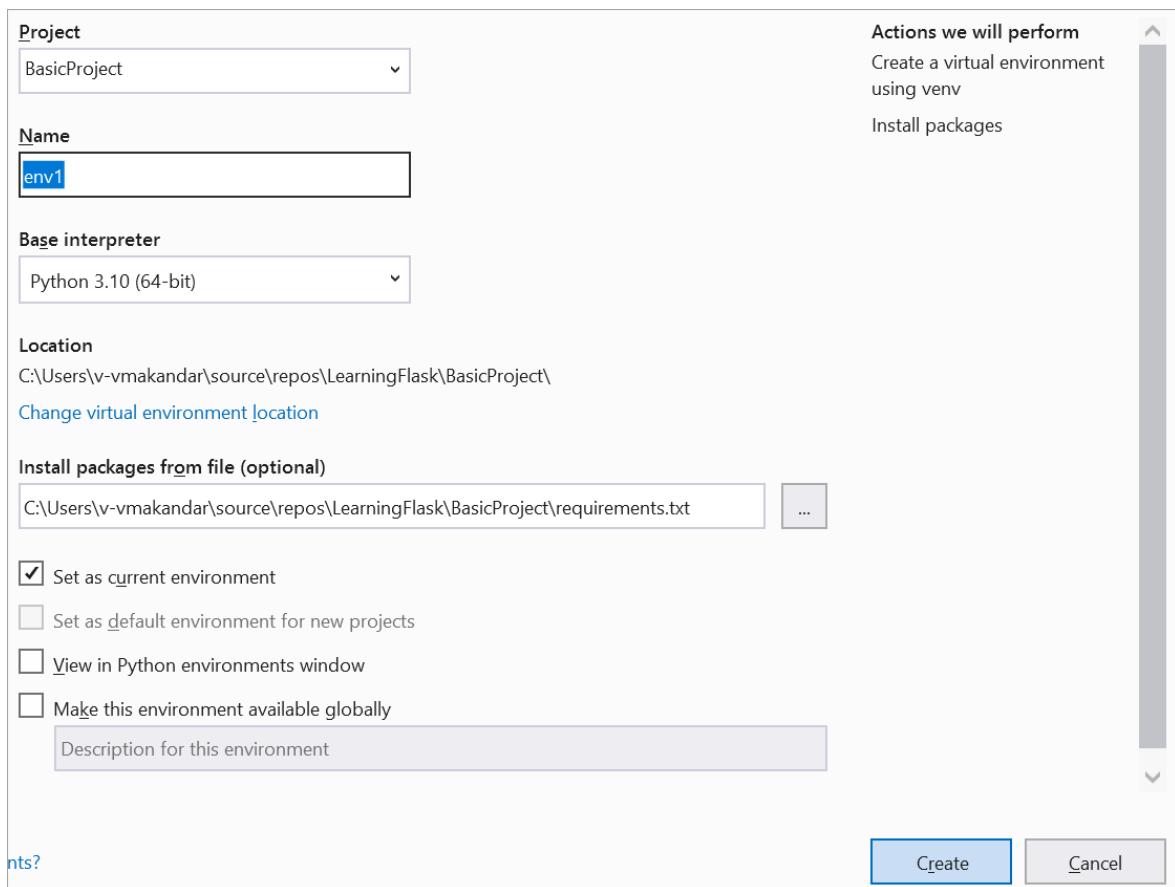
Etapa 1-3: Criar o ambiente virtual e excluí-lo do controle do código-fonte

Agora que você configurou o controle do código-fonte para o projeto, é possível criar no ambiente virtual os pacotes necessários do Flask que o projeto exige. Você pode usar o **Team Explorer** para excluir a pasta do ambiente do controle do código-fonte.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Ambientes do Python** e selecione **Adicionar Ambiente**.



2. Selecione **Criar** para aceitar os padrões na caixa de diálogo Adicionar Ambiente Virtual. (Se desejar, o nome do ambiente virtual pode ser alterado. Essa ação alterará apenas o nome da subpasta, mas `env` é uma convenção padrão).



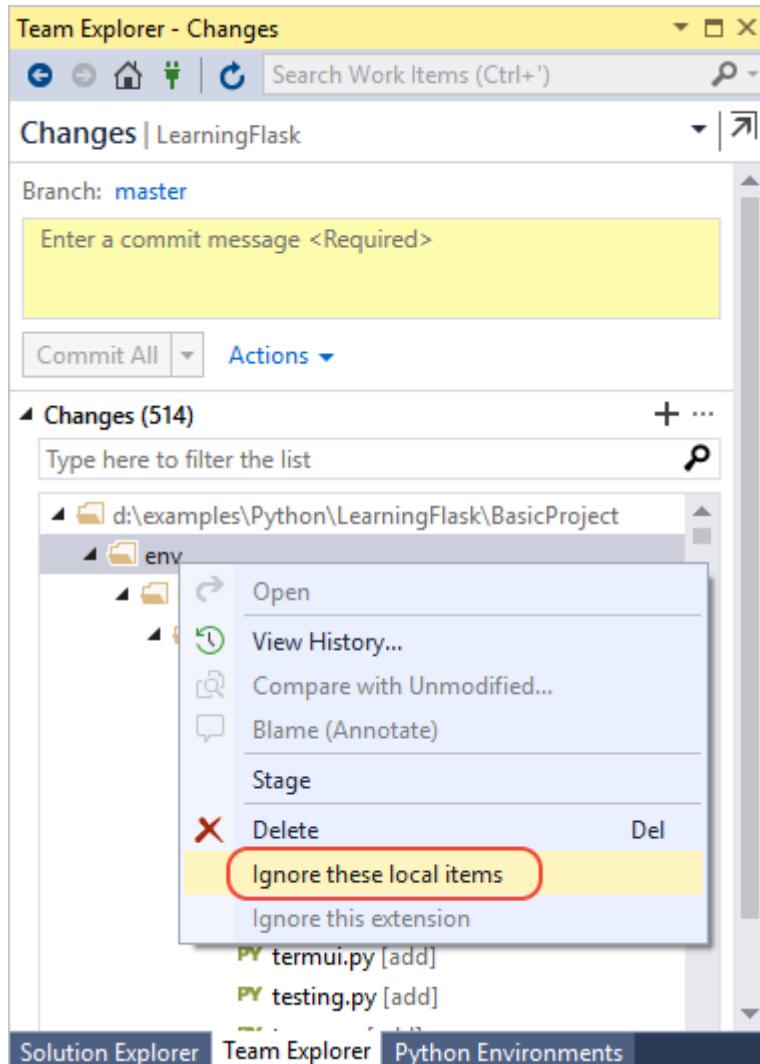
3. Dê a concessão aos privilégios de administrador, se solicitado, e aguarde alguns minutos enquanto o Visual Studio baixa e instala os pacotes. Durante esse tempo,

milhares de arquivos serão transferidos para diversas subpastas. Você pode ver o progresso na janela **Saída** no Visual Studio. Enquanto você aguarda, analise as seções de perguntas a seguir.

4. Nos controles do Git do Visual Studio (na barra de status), selecione o indicador de alterações (que mostra **99***) que abre a página **Alterações** no **Team Explorer**.

A criação do ambiente virtual apresentou milhares de alterações, mas nenhuma delas precisará ser incluída no controle do código-fonte já que você (ou qualquer outra pessoa que venha a clonar o projeto) poderá sempre recriar o ambiente com base em *requirements.txt*.

Para excluir o ambiente virtual, clique com o botão direito do mouse na pasta **env** e selecione **Ignorar estes itens locais**.



5. Depois de excluir o ambiente virtual, as únicas alterações restantes são as referentes ao arquivo de projeto e a *.gitignore*. O arquivo *.gitignore* contém uma entrada adicional para a pasta do ambiente virtual. Você pode clicar duas vezes no arquivo para ver uma comparação.

6. Digite uma mensagem de confirmação, escolha o botão **Confirmar Todos** e envie as confirmações por push para o repositório remoto.

Pergunta: Por que criar um ambiente virtual?

Resposta: Um ambiente virtual é uma ótima maneira de isolar as dependências exatas do seu aplicativo. Esse isolamento evita conflitos em um ambiente global do Python e auxilia nos testes e na colaboração. Com o tempo, à medida que desenvolver um aplicativo, invariavelmente, você introduzirá muitos pacotes úteis do Python. Mantendo os pacotes em um ambiente virtual específico do projeto, você pode atualizar com facilidade o arquivo *requirements.txt* do projeto que descreve esse ambiente, incluído no controle do código-fonte. Quando o projeto é copiado para outros computadores, incluindo servidores de build, servidores de implantação e outros computadores de desenvolvimento, é fácil recriar o ambiente usando apenas o *requirements.txt* (é por isso que o ambiente não precisa estar no controle do código-fonte). Para obter mais informações, confira [Usar ambientes virtuais](#).

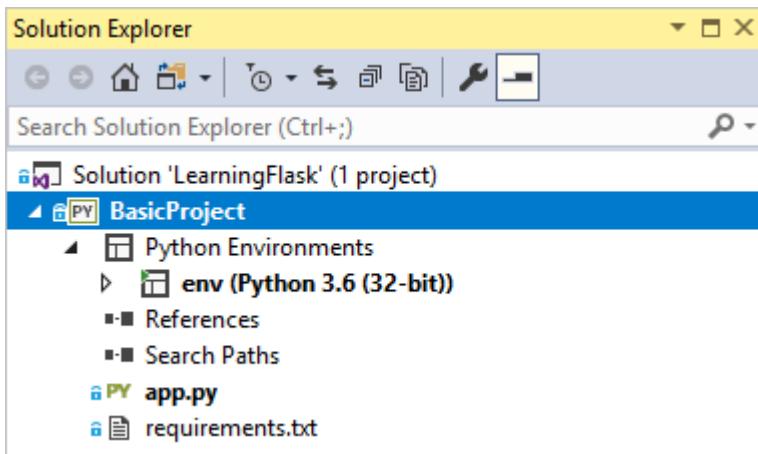
Pergunta: Como faço para remover um ambiente virtual que já está confirmado no controle do código-fonte?

Resposta: Primeiro, edite o arquivo *.gitignore* para excluir a pasta: localize a seção ao final com o comentário `# Python Tools for Visual Studio (PTVS)` e adicione uma nova linha à pasta do ambiente virtual, como `/BasicProject/env`. (Como o Visual Studio não mostra o arquivo no **Gerenciador de Soluções**, abra-o diretamente usando os comandos de menu **Arquivo > Abrir > Arquivo**. Você também pode abrir o arquivo no **Team Explorer**: na página **Configurações**, selecione **Configurações do Repositório**, navegue até a seção **Ignorar & Arquivos de Atributos** e selecione o link **Editar** próximo a *.gitignore*).

Em segundo lugar, abra uma janela Comando, navegue para a pasta, como *BasicProject*, que contém a pasta do ambiente virtual, como *env*, e execute `git rm -r env`. Em seguida, confirme essas alterações na linha de comando (`git commit -m 'Remove venv'`) ou confirme na página **Alterações** do **Team Explorer**.

Etapa 1-4: Examinar o código de texto clichê

1. Quando a criação do projeto for concluída, você verá a solução e o projeto no **Gerenciador de Soluções**, em que o projeto contém apenas dois arquivos, *app.py* e *requirements.txt*:



2. Conforme observado anteriormente, o arquivo *requirements.txt* especifica a dependência de pacote do Flask. A presença desse arquivo é que faz com que você seja convidado a criar um ambiente virtual ao desenvolver o projeto pela primeira vez.
3. O único arquivo *app.py* contém três partes. A primeira é uma instrução `import` para o Flask, criando uma instância da classe `Flask`, atribuída à variável `app` e então atribuindo uma variável `wsgi_app` (útil ao implantar um host da Web, mas não será usada no momento):

Python

```
from flask import Flask
app = Flask(__name__)

# Make the WSGI interface available at the top level so wfastcgi can
# get it.
wsgi_app = app.wsgi_app
```

4. A segunda parte, ao fim do arquivo, é um trecho do código opcional que inicia o servidor de desenvolvimento do Flask com valores de porta e de host específicos extraídos de variáveis de ambiente (padrão para localhost:5555):

Python

```
if __name__ == '__main__':
    import os
    HOST = os.environ.get('SERVER_HOST', 'localhost')
    try:
        PORT = int(os.environ.get('SERVER_PORT', '5555'))
    except ValueError:
        PORT = 5555
    app.run(HOST, PORT)
```

5. A terceira é um breve trecho de código que atribui uma função a uma rota de URL, o que significa que a função oferece o recurso identificado pela URL. Defina rotas usando o decorador `@app.route` do Flask, cujo argumento é a URL relativa da raiz do site. Como é possível ver no código, a função retorna apenas uma cadeia de caracteres de texto, que é suficiente para um navegador renderizar. Nas etapas seguintes, você renderizará páginas mais avançadas com HTML.

Python

```
@app.route('/')
def hello():
    """Renders a sample page."""
    return "Hello World!"
```

Pergunta: Qual é a finalidade do argumento `name` para a classe do Flask?

Resposta: o argumento é o nome do módulo ou do pacote do aplicativo e informa ao Flask onde procurar modelos, arquivos estáticos e outros recursos que pertencem ao aplicativo. Para aplicativos contidos em um único módulo, `__name__` é sempre o valor adequado. Também é importante para as extensões que precisam de informações de depuração. Para obter mais informações, e outros argumentos, confira a [documentação de classes do Flask](#) (flask.pocoo.org).

Pergunta: Uma função pode ter mais de um decorador de rota?

Resposta: sim, será possível usar quantos decoradores você quiser se a mesma função servir para várias rotas. Por exemplo, para usar a função `hello` para "/" e "/hello", use o seguinte código:

Python

```
@app.route('/')
@app.route('/hello')
def hello():
    """Renders a sample page."""
    return "Hello World!"
```

Pergunta: Como o Flask trabalha com rotas de URL de variável e parâmetros de consulta?

Resposta: em uma rota, marque qualquer variável com `<variable_name>`, e o Flask passará a variável para a função usando um argumento nomeado no caminho da URL. Por exemplo, uma rota na forma de `/hello/<name>` gera o argumento de cadeia de caracteres `name` para a função. Os parâmetros de consulta também estão disponíveis por meio da propriedade `request.args`, especificamente por meio do método `request.args.get`.

Python

```
# URL: /hello/<name>?message=Have%20a%20nice%20day
@app.route('/hello/<name>')
def hello(name):
    msg = request.args.get('message', '')
    return "Hello " + name + "!" + msg + "."
```

Para alterar o tipo, insira `int`, `float`, `path` à frente da variável (que aceita barras para delinear nomes de pasta) e `uuid`. Para obter detalhes, confira [Regras de variáveis](#) na documentação do Flask.

Pergunta: O Visual Studio pode gerar um arquivo requirements.txt a partir de um ambiente virtual depois de instalar outros pacotes?

Resposta: Sim. Expanda o nó **Ambientes do Python**, clique com o botão direito do mouse no ambiente virtual e escolha o comando **Gerar requirements.txt**. É recomendável usar esse comando periodicamente conforme você modifica o ambiente e confirma as alterações em `requirements.txt` no controle do código-fonte, juntamente com outras alterações de código que dependem desse ambiente. Se você configurar a integração contínua em um servidor de compilação, deverá gerar o arquivo e confirmar as alterações sempre que modificar o ambiente.

Etapa 1-5: Executar o projeto

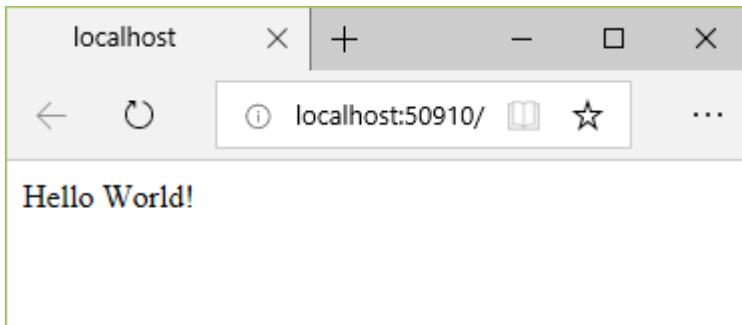
1. No Visual Studio, selecione **Depurar>Iniciar Depuração (F5)** ou use o botão **Servidor Web** na barra de ferramentas (o navegador que você vai ver pode variar):



2. Qualquer comando atribui um número da porta aleatório à variável de ambiente PORT e, em seguida, executa `python app.py`. O código inicia o aplicativo usando essa porta dentro do servidor de desenvolvimento do Flask. Se o Visual Studio

informar Falha ao iniciar o depurador com uma mensagem que alerta para a ausência de um arquivo de inicialização, clique com o botão direito do mouse em **app.py** no **Gerenciador de Soluções** e selecione **Definir como Arquivo de Inicialização**.

3. Ao iniciar o servidor, você verá que a janela do console aberta exibe o log do servidor. O Visual Studio abre automaticamente um navegador para `http://localhost:<port>`, em que você verá a mensagem renderizada pela função `hello`:



4. Quando terminar, interrompa o servidor fechando a janela do console ou usando o comando **Depurar>Parar Depuração** no Visual Studio.

Pergunta: Qual é a diferença entre usar os comandos do menu Depuração e os comandos do servidor no submenu do Python do projeto?

Resposta: Além dos comandos do menu **Depurar** e dos botões da barra de ferramentas, você também pode iniciar o servidor usando os comandos **Python>Executar servidor** ou **Python>Executar o servidor de depuração** no menu de contexto do projeto. Os dois comandos abrem uma janela de console na qual você vê a URL local (`localhost:port`) do servidor em execução. No entanto, você deve abrir manualmente um navegador usando essa URL, já que a execução do servidor de depuração não inicia automaticamente o depurador do Visual Studio. Se desejar, você pode posteriormente anexar um depurador ao processo em execução usando o comando **Depurar>Anexar ao Processo**.

Próximas etapas

Neste ponto, o projeto básico do Flask contém o código de inicialização e o código de página no mesmo arquivo. É melhor separar essas duas questões e também separar o HTML e os dados usando modelos.

[Criar um aplicativo do Flask com modos de exibição e modelos de página](#)

Etapa 2: Criar um aplicativo Flask com exibições e modelos de página

Artigo • 08/09/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Criar uma solução e um projeto do Visual Studio](#)

Na etapa 1 deste tutorial, você criou um aplicativo Flask com uma página e todo o código em um único arquivo. Para permitir o desenvolvimento futuro, é melhor refatorar o código e criar uma estrutura para modelos de página. Em particular, separe o código para exibições do aplicativo de outros aspectos como o código de inicialização.

Nesta etapa, você aprenderá a:

- ✓ Refatorar o código do aplicativo para separar exibições do código de inicialização (etapa 2-1)
- ✓ Renderizar uma exibição usando um modelo de página (etapa 2-2)

Etapa 2-1: Refatorar o projeto para dar suporte ao desenvolvimento adicional

No código criado pelo modelo "Projeto Web em Branco do Flask", você tem um único arquivo `app.py` que contém o código de inicialização junto com uma única exibição. Para permitir o desenvolvimento adicional de um aplicativo com várias exibições e modelos, é melhor separar essas preocupações.

1. Na pasta do seu projeto, crie uma pasta de aplicativo chamada `HelloFlask` (clique com o botão direito do mouse no projeto em **Gerenciador de Soluções** e selecione **Adicionar>Nova Pasta**.)
2. Na pasta `HelloFlask`, crie um arquivo chamado `_init_.py` com o seguinte conteúdo que cria a instância `Flask` e carrega as exibições do aplicativo (criadas na próxima etapa):

Python

```
from flask import Flask
app = Flask(__name__)

import HelloFlask.views
```

3. Na pasta *HelloFlask*, crie um arquivo chamado *views.py* com o conteúdo a seguir.

O nome *views.py* é importante, porque você usou `import HelloFlask.views` dentro de `__init__.py`; você verá um erro em tempo de execução se os nomes não corresponderem.

Python

```
from flask import Flask
from HelloFlask import app

@app.route('/')
@app.route('/home')
def home():
    return "Hello Flask!"
```

Além de renomear a função e a rota como `home`, esse código contém o código de renderização da página de *app.py* e importa o objeto `app` declarado em *__init__.py*.

4. Crie uma subpasta em *HelloFlask* chamada *templates*, que permanecerá vazia por enquanto.

5. Na pasta raiz do projeto, renomeie *app.py* como *runserver.py* e faça o conteúdo corresponder ao seguinte código:

Python

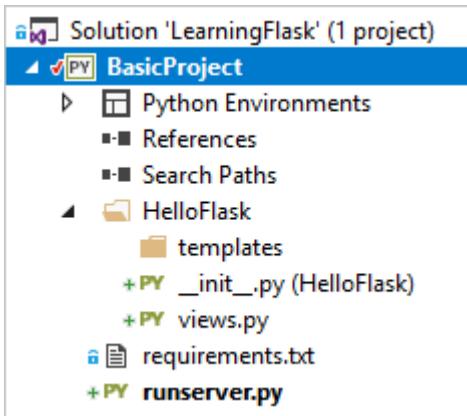
```
import os
from HelloFlask import app      # Imports the code from
HelloFlask/__init__.py

if __name__ == '__main__':
    HOST = os.environ.get('SERVER_HOST', 'localhost')

    try:
        PORT = int(os.environ.get('SERVER_PORT', '5555'))
    except ValueError:
        PORT = 5555

    app.run(HOST, PORT)
```

6. A estrutura do seu projeto deve ser semelhante à da imagem a seguir:



7. Selecione **Depurar>Iniciar Depuração (F5)** ou use o botão **Servidor Web** na barra de ferramentas (o navegador exibido poderá variar) para iniciar o aplicativo e abrir um navegador. Experimente as rotas de URL / e /home.
8. Também é possível definir pontos de interrupção em várias partes do código e reiniciar o aplicativo para seguir a sequência de inicialização. Por exemplo, defina um ponto de interrupção nas primeiras linhas de `runserver.py` e `HelloFlask_*init_.py` e na linha `return "Hello Flask!"` de `views.py`. Em seguida, reinicie o aplicativo (**Depurar>Reiniciar, Ctrl+Shift+F5** ou o botão de barra de ferramentas mostrado abaixo) e execute o código em etapas (**F10**) ou execute-o em cada ponto de interrupção usando **F5**.

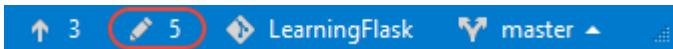


9. Interrompa o aplicativo ao terminar.

Confirmar o controle do código-fonte

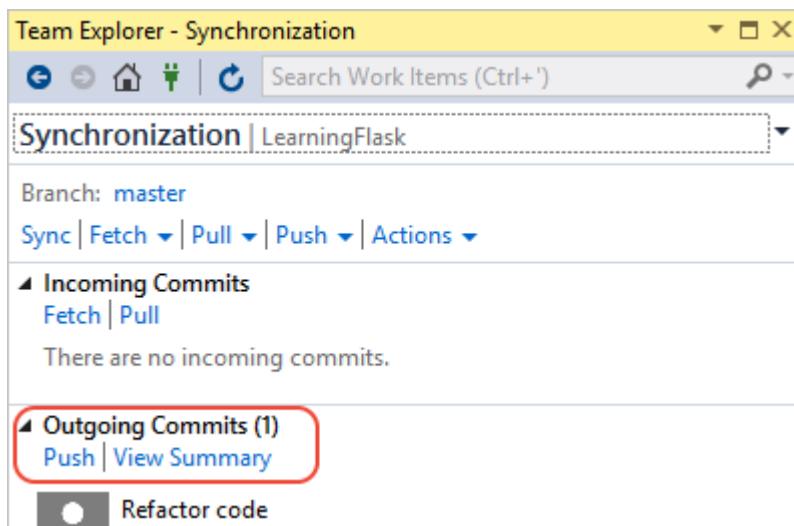
Depois de fazer as alterações no código e testar com êxito, você pode examinar e confirmar as alterações no controle do código-fonte. Em etapas posteriores, quando este tutorial lembrar você de fazer commit com o controle do código-fonte novamente, consulte esta seção.

1. Selecione o botão de alterações na parte inferior do Visual Studio (circulado abaixo), que encaminha para o **Team Explorer**.



2. No **Team Explorer**, digite uma mensagem de confirmação como "Refatorar código" e selecione **Confirmar Tudo**. Quando a confirmação for concluída, você verá uma mensagem **Commit <hash> criada localmente. Sincronize para compartilhar suas alterações com o servidor**. Se você quiser enviar alterações por push para o repositório remoto, selecione **Sincronizar** e, em seguida, selecione

efetuar push em **Commits de saída**. Também é possível acumular várias confirmações locais antes de enviar para o repositório remoto.



Pergunta: Com que frequência é necessário fazer a confirmação no controle do código-fonte?

Resposta: A confirmação de alterações no controle do código-fonte cria um registro no log de alterações e um ponto em que é possível reverter o repositório, se necessário. Cada confirmação também pode ser examinada para suas alterações específicas. Como as confirmações no GIT não são caras, é melhor realizar confirmações frequentes do que acumular um grande número de alterações em uma única confirmação. Não é necessário confirmar cada pequena alteração em arquivos individuais. Normalmente, você realiza uma confirmação ao adicionar uma funcionalidade, alterando uma estrutura como você fez nesta etapa, ou refatorar algum código. Verifique também com outras pessoas da sua equipe para a granularidade de confirmações que funcionam melhor para todos.

A frequência de confirmação e a frequência de envio de confirmações por push a um repositório remoto são duas preocupações diferentes. Você pode acumular várias confirmações no seu repositório local antes de enviá-las por push para o repositório remoto. A frequência dos commits depende de como sua equipe deseja gerenciar o repositório.

Etapa 2-2: Usar um modelo para renderizar uma página

A função `home` que você tem até agora em `views.py` não gera nada além de uma resposta HTTP de texto sem formatação para a página. No entanto, a maioria das páginas da Web reais respondem com páginas HTML avançadas que incorporam dados

dinâmicos com frequência. A principal razão para definir uma exibição usando uma função é gerar esse conteúdo dinamicamente.

Como o valor retornado para a exibição é apenas uma cadeia de caracteres, é possível criar qualquer HTML que desejar dentro de uma cadeia de caracteres, usando conteúdo dinâmico. No entanto, como é melhor separar a marcação dos dados, é melhor inserir a marcação em um modelo e manter os dados no código.

1. Para iniciantes, edite `views.py` para que ela contenha o seguinte código que usa HTML embutido para a página com algum conteúdo dinâmico:

Python

```
from datetime import datetime
from flask import render_template
from HelloFlask import app

@app.route('/')
@app.route('/home')
def home():
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    html_content = "<html><head><title>Hello Flask</title></head>
<body>"
    html_content += "<strong>Hello Flask!</strong> on " + formatted_now
    html_content += "</body></html>

    return html_content
```

2. Execute o aplicativo e atualize a página poucas vezes para ver que a data/hora está atualizada. Interrompa o aplicativo ao terminar.
3. Para converter a renderização da página para usar um modelo, crie um arquivo chamado `index.html` na pasta `templates` com o seguinte conteúdo, em que `{{ content }}` é um espaço reservado ou um token de substituição (também chamado de uma *variável de modelo*) para o qual você fornece um valor no código:

HTML

```
<html>
  <head>
    <title>Hello Flask</title>
  </head>

  <body>
    {{ content }}
```

```
</body>
</html>
```

4. Modifique a função `home` para usar `render_template` para carregar o modelo e fornecer um valor para "conteúdo", que é feito usando um argumento nomeado correspondente ao nome do espaço reservado. O Flask procura automaticamente os modelos na pasta `templates`; portanto o caminho para o modelo é relativo a essa pasta:

Python

```
def home():
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    return render_template(
        "index.html",
        content = "<strong>Hello, Flask!</strong> on " + formatted_now)
```

5. Execute o aplicativo e veja os resultados. Observe que o HTML embutido no valor `content` não é renderizado como HTML, porque o mecanismo de modelagem (Jinja) ignora automaticamente o conteúdo HTML. O escape automático impede vulnerabilidades accidentais a ataques de injeção. Os desenvolvedores geralmente coletam entradas de uma página e a usam como um valor em outra por meio de um espaço reservado de modelo. O escape também funciona como um lembrete de que é melhor manter o HTML fora do código.

Da mesma forma, examine `templates\index.html` para conter espaços reservados distintos em cada parte dos dados dentro da marcação:

HTML

```
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <strong>{{ message }}</strong>{{ content }}
  </body>
</html>
```

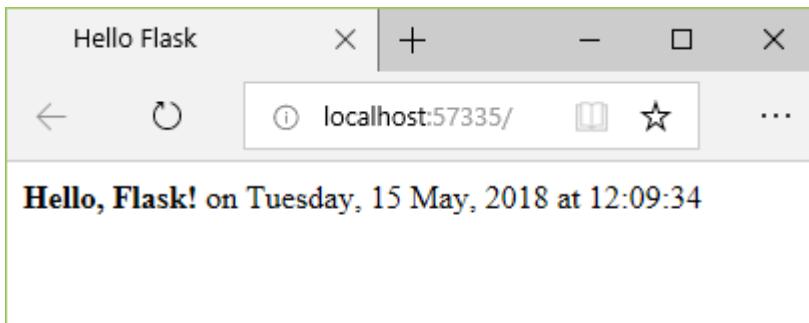
Em seguida, atualize a função `home` para fornecer valores para todos os espaços reservados:

Python

```
def home():
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    return render_template(
        "index.html",
        title = "Hello Flask",
        message = "Hello, Flask!",
        content = " on " + formatted_now)
```

6. Execute o aplicativo novamente e veja a saída renderizada corretamente.



7. Você pode confirmar suas alterações no controle do código-fonte e atualizar seu repositório remoto. Para obter mais informações, consulte a [etapa 2-1](#).

Pergunta: Os modelos de página precisam ficar em um arquivo separado?

Resposta: Embora os modelos geralmente sejam mantidos em arquivos HTML separados, também é possível usar um modelo embutido. É recomendável usar arquivos separados para manter uma separação clara entre o markup e o código.

Pergunta: Os modelos precisam usar a extensão de arquivo .html?

Resposta: A extensão *.html* dos arquivos de modelo de página é totalmente opcional, porque você sempre pode identificar o caminho relativo exato para o arquivo no primeiro argumento para a função `render_template`. No entanto, o Visual Studio (e outros editores) costuma fornecer funcionalidades como preenchimento de código e coloração de sintaxe com arquivos *.html*, o que supera o fato de os modelos de página não serem HTML.

Quando você está trabalhando com um projeto do Flask, o Visual Studio detecta automaticamente quando o arquivo HTML que você está editando é, na verdade, um modelo do Flask e fornece algumas funcionalidades de preenchimento automático. Por

exemplo, quando você começa a digitar um comentário no modelo de página do Flask, `{#`, o Visual Studio fornece automaticamente os caracteres de fechamento `#}`. Os comandos **Comentar Seleção** e **Remover Marca de Comentário da Seleção** (no menu **Editar>Avançado** e na barra de ferramentas) também usam comentários de modelo em vez de comentários em HTML.

Pergunta: Os modelos podem ser organizados em subpastas adicionais?

Resposta: Sim. É possível usar subpastas e, em seguida, referir-se ao caminho relativo em *templates* nas chamadas a `render_template`. Isso é uma ótima maneira de criar efetivamente namespaces para os modelos.

Próximas etapas

Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo

Aprofunde-se um pouco mais

- Início rápido do Flask – Renderizando modelos ↗ (flask.pocoo.org)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-flask](https://github.com/Microsoft/python-sample-vs-learning-flask) ↗

Etapa 3: Fornecer arquivos estáticos, adicionar páginas e usar a herança do modelo com o aplicativo Flask

Artigo • 19/06/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Etapa anterior: [Criar um aplicativo do Flask com modos de exibição e modelos de página](#)

Nas etapas anteriores deste tutorial, você aprendeu como criar um aplicativo mínimo do Flask com uma única página HTML autocontido. Os aplicativos web modernos geralmente são compostos por várias páginas e usam recursos compartilhados, como arquivos CSS e JavaScript, para fornecer comportamento e estilo consistentes.

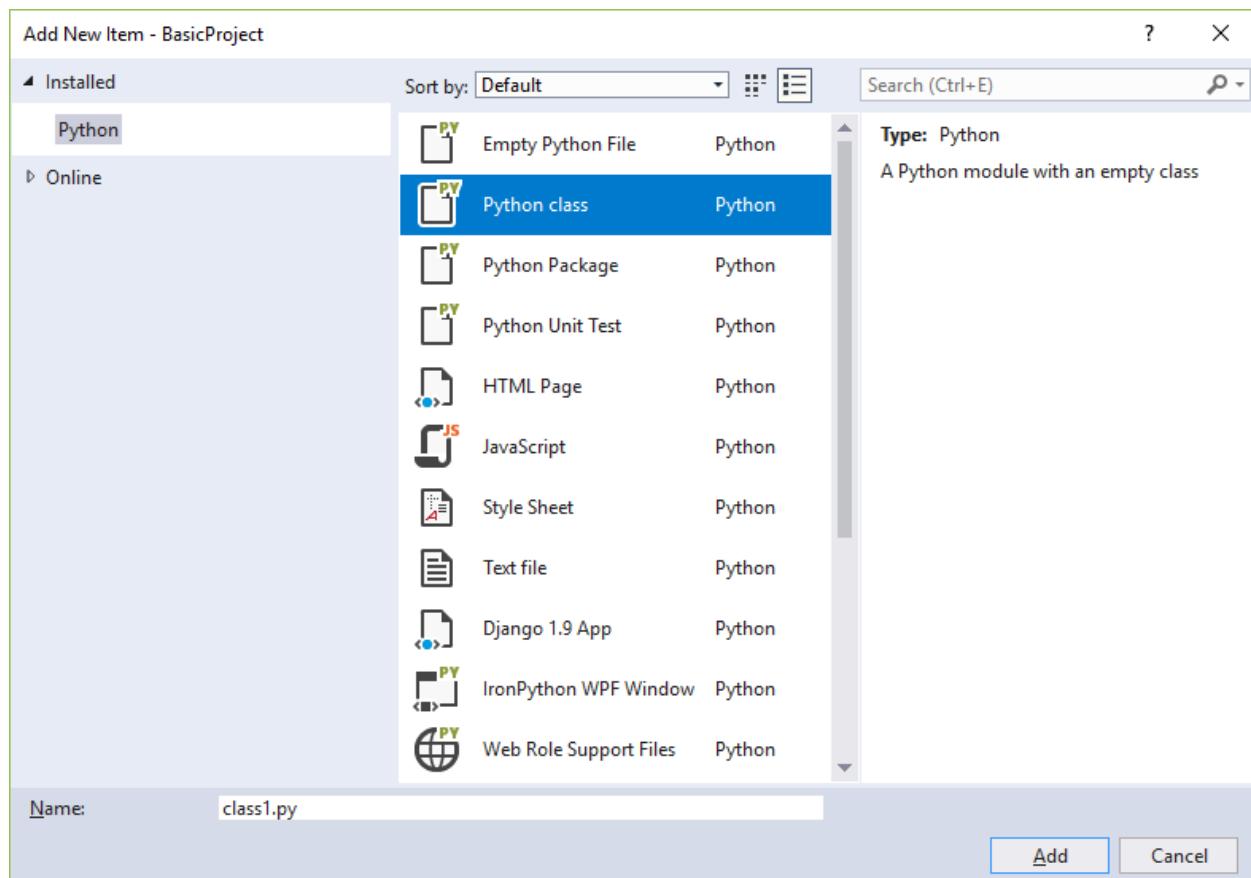
Nesta etapa, você aprenderá a:

- ✓ Usar modelos de item do Visual Studio para adicionar rapidamente novos arquivos de diferentes tipos com código clichê conveniente (etapa 3-1)
- ✓ Fornecer arquivos estáticos do código (etapa 3-2, opcional)
- ✓ Adicionar mais páginas ao aplicativo (etapa 3-3)
- ✓ Usar a herança do modelo para criar um cabeçalho e uma barra de navegação usados nas páginas (etapa 3-4)

Etapa 3-1: Familiarizar-se com os modelos de item

À medida que desenvolve um aplicativo do Flask, geralmente você adiciona vários outros arquivos Python, HTML, CSS e JavaScript. Para cada tipo de arquivo (bem como para outros arquivos, como *web.config*, que podem ser necessários para a implantação), o Visual Studio fornece [modelos de itens](#) convenientes para você começar.

Para ver os modelos disponíveis, vá para **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta em que você deseja criar o item, selecione **Adicionar>Novo Item**:



Para usar um modelo, selecione o modelo desejado, especifique um nome para o arquivo e selecione **OK**. A adição de um item dessa maneira adiciona automaticamente o arquivo ao projeto do Visual Studio e marca as alterações para controle do código-fonte.

Pergunta: Como o Visual Studio sabe quais modelos de item oferecer?

Resposta: O arquivo de projeto do Visual Studio (`.pyproj`) contém um identificador de tipo de projeto que o marca como um projeto do Python. O Visual Studio usa esse identificador de tipo para mostrar apenas os modelos de item adequados para o tipo de projeto. Dessa forma, o Visual Studio pode fornecer um conjunto avançado de modelos de item para muitos tipos de projeto sem solicitar que você os classifique toda vez.

Etapa 3-2: Fornecer arquivos estáticos do seu aplicativo

Em um aplicativo Web criado com Python (usando qualquer estrutura), seus arquivos em Python sempre são executados no servidor do host da Web e nunca são transmitidos para o computador de um usuário. Mas outros arquivos, como CSS e JavaScript, são usados exclusivamente pelo navegador, de modo que o servidor do host

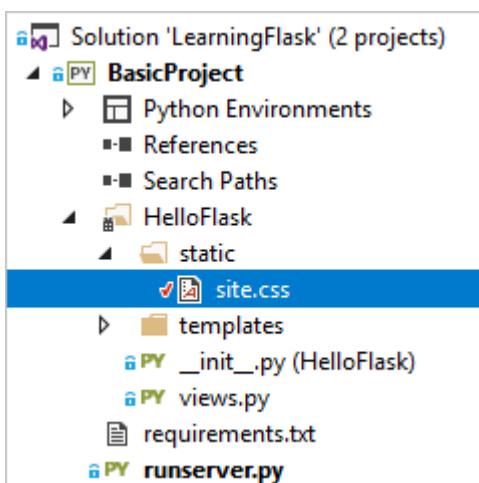
simplesmente os entrega sem alterações sempre que são solicitados. Esses arquivos são chamados de "estáticos", e o Flask pode fornecê-los automaticamente sem que você precise escrever código. Dentro de arquivos HTML, por exemplo, é possível referenciar arquivos estáticos usando um caminho relativo no projeto. A primeira seção desta etapa adiciona um arquivo CSS ao modelo de página existente.

Quando você precisar entregar um arquivo estático do código, como por meio de uma implementação de ponto de extremidade de API, o Flask fornecerá um método conveniente que permite a você referenciar arquivos usando caminhos relativos dentro de uma pasta chamada `static` (na raiz do projeto). A segunda seção desta etapa demonstra esse método usando um arquivo de dados estático simples.

Em qualquer caso, você pode organizar os arquivos em `static` como desejar.

Usar um arquivo estático em um modelo

1. No Gerenciador de Soluções, clique com o botão direito do mouse na pasta **HelloFlask** no projeto do Visual Studio, selecione **Adicionar>Nova pasta** e nomeie a pasta `static`.
2. Clique com o botão direito do mouse na pasta `static` e selecione **Adicionar>Novo item**. Na caixa de diálogo exibida, selecione o modelo **Folha de estilos**, nomeie o arquivo `site.css` e selecione **OK**. O arquivo `site.css` é exibido no projeto e aberto no editor. A estrutura de pastas deve ser semelhante à imagem a seguir:



3. Substitua o conteúdo de `site.css` pelo seguinte código e salve o arquivo:

```
css

.message {
    font-weight: 600;
    color: blue;
}
```

4. Substitua o conteúdo do arquivo `templates/index.html` do aplicativo pelo código a seguir, que substitui o elemento `` usado na etapa 2 por um `` que referencia a classe de estilo `message`. O uso de uma classe de estilo dessa maneira oferece muito mais flexibilidade ao estilo do elemento.

HTML

```
<html>
  <head>
    <title>{{ title }}</title>
    <link rel="stylesheet" type="text/css" href="/static/site.css"
  />
  </head>
  <body>
    <span class="message">{{ message }}</span>{{ content }}
  </body>
</html>
```

5. Execute o projeto para observar os resultados. Interrompa o aplicativo quando terminar e pode confirmar as alterações no controle do código-fonte se desejar (conforme explicado na [etapa 2](#)).

Fornecer um arquivo estático do código

O Flask fornece uma função chamada `serve_static_file` que você pode chamar por meio do código para referenciar qualquer arquivo dentro da pasta `static` do projeto. O processo a seguir cria um ponto de extremidade de API simples que retorna um arquivo de dados estáticos.

1. Se você ainda não fez isto, crie uma pasta `static`: no Gerenciador de Soluções, clique com o botão direito do mouse na pasta `HelloFlask` no projeto do Visual Studio, selecione **Adicionar>Nova pasta** e nomeie a pasta `static`.
2. Na pasta `static`, crie um arquivo de dados JSON estático chamado `data.json` com o seguinte conteúdo (dados de exemplo sem sentido):

JSON

```
{
  "01": {
    "note" : "Data is very simple because we're demonstrating only
the mechanism."
  }
}
```

3. Em `views.py`, adicione uma função com a rota `/api/data` que retorna o arquivo de dados estático usando o método `send_static_file`:

Python

```
@app.route('/api/data')
def get_data():
    return app.send_static_file('data.json')
```

4. Execute o aplicativo e navegue até o ponto de extremidade `/api/data` para ver se o arquivo estático será retornado. Interrompa o aplicativo ao terminar.

Pergunta: Há convenções para organizar arquivos estáticos?

Resposta: Você pode adicionar outros arquivos CSS, JavaScript e HTML à sua pasta `static` como quiser. Uma maneira comum de organizar arquivos estáticos é criar subpastas chamadas `fonts`, `scripts` e `content` (para folhas de estilo e outros arquivos).

Pergunta: Como fazer para manipular variáveis de URL e parâmetros de consulta em uma API?

Resposta: Confira a resposta na etapa 1-4 para a [Pergunta: Como o Flask trabalha com rotas de URL de variável e parâmetros de consulta?](#)

Etapa 3-3: Adicionar uma página ao aplicativo

A adição de outra página ao aplicativo significa o seguinte:

- Adicione uma função em Python que defina o modo de exibição.
- Adicione um modelo para a marcação da página.
- Adicione o roteamento necessário ao arquivo `urls.py` do projeto do Flask.

As etapas a seguir adicionam uma página "Sobre" ao projeto "HelloFlask" e links para essa página na home page:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse na pasta `templates`, selecione **Adicionar>Novo item**, selecione o modelo de item **Página HTML**, nomeie o arquivo `about.html` e selecione **OK**.

 Dica

Se o comando **Novo Item** não for exibido no menu **Adicionar**, verifique se você interrompeu o aplicativo para que o Visual Studio saia do modo de depuração.

2. Substitua o conteúdo de *about.html* pela seguinte marcação (substitua o link explícito para a home page por uma barra de navegação simples na etapa 3-4):

```
HTML

<html>
  <head>
    <title>{{ title }}</title>
    <link rel="stylesheet" type="text/css" href="/static/site.css"
  />
  </head>
  <body>
    <div><a href="home">Home</a></div>
    {{ content }}
  </body>
</html>
```

3. Abra o arquivo *views.py* do aplicativo e adicione uma função chamada `about` que usa o modelo:

```
Python

@app.route('/about')
def about():
    return render_template(
        "about.html",
        title = "About HelloFlask",
        content = "Example app page for Flask.")
```

4. Abra o arquivo *templates/index.html* e adicione a seguinte linha imediatamente dentro do elemento `<body>` para criar um link para a página About (novamente, substitua esse link por uma barra de navegação na etapa 3-4):

```
HTML

<div><a href="about">About</a></div>
```

5. Salve todos os arquivos usando o comando de menu **Arquivo>Salvar Tudo** ou pressione **Ctrl+Shift+S**. (Essa etapa não é necessária, pois a execução do projeto no Visual Studio salva os arquivos automaticamente. No entanto, é um bom comando para se saber!)

6. Execute o projeto para observar os resultados e verificar a navegação entre as páginas. Interrompa o aplicativo quando terminar.

Pergunta: o nome de uma função de página é importante para o Flask?

Resposta: não, porque ele é o decorador `@app.route` que determina as URLs para as quais o Flask chama a função para gerar uma resposta. Normalmente, os desenvolvedores correspondem o nome da função à rota, mas essa correspondência não é obrigatória.

Etapa 3-4: Usar a herança do modelo para criar um cabeçalho e uma barra de navegação

Em vez de ter links de navegação explícitos em cada página, os aplicativos Web modernos normalmente usam um cabeçalho de marca e uma barra de navegação que fornece os links de página mais importantes, menus pop-up e assim por diante. Para garantir que o aplicativo seja consistente, o cabeçalho e a barra de navegação devem ser os mesmos em todas as páginas, sem repetir o mesmo código em cada modelo de página. Em vez disso, defina as partes comuns de todas as páginas em um único local.

O sistema de modelos do Flask (Jinja por padrão) fornece dois meios para reutilização de elementos específicos em vários modelos: inclui e herança.

- *Includes* corresponde a outros modelos de página que você insere em um local específico no modelo de referência usando a sintaxe `{% include <template_path>%}`. Você também pode usar uma variável se quiser alterar o caminho dinamicamente no código. Os elementos Inclui normalmente são usados no corpo de uma página para extrair o modelo compartilhado em um local específico na página.
- *Inheritance* usa o `{% extends <template_path>%}` no início de um modelo de página para especificar um modelo de base compartilhado no qual o modelo de referência se baseará. O elemento Herança costuma ser usado para definir um layout compartilhado, uma barra de navegação e outras estruturas para as páginas de um aplicativo, de modo que os modelos de referência precisam apenas adicionar ou modificar áreas específicas do modelo de base chamadas *blocos*.

Em ambos os casos, `<template_path>` é relativo à pasta *templates* do aplicativo (`../` ou `./` também é permitido).

Um modelo de base delineia *blocos* usando as marcações `{% block <block_name> %}` e `{% endblock %}`. Se um modelo de referência usar marcações com o mesmo nome de bloco, o conteúdo do bloco substituirá o do modelo de base.

As etapas a seguir demonstram a herança:

1. Na pasta *templates* do aplicativo, crie um arquivo HTML (usando o menu de contexto **Adicionar>Novo item** ou **Adicionar>Página HTML**) chamado *layout.html* e substitua o conteúdo pela marcação abaixo. Veja que esse modelo contém um bloco chamado "conteúdo", que representa tudo o que as páginas de referência precisam substituir:

```
HTML

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>{{ title }}</title>
    <link rel="stylesheet" type="text/css" href="/static/site.css" />
</head>

<body>
    <div class="navbar">
        <a href="/" class="navbar-brand">Hello Flask</a>
        <a href="{{ url_for('home') }}" class="navbar-item">Home</a>
        <a href="{{ url_for('about') }}" class="navbar-item">About</a>
    </div>

    <div class="body-content">
        {% block content %}
        {% endblock %}
        <hr/>
        <footer>
            <p>&copy; 2018</p>
        </footer>
    </div>
</body>
</html>
```

2. Adicione os seguintes estilos ao arquivo *static/site.css* do aplicativo (este passo a passo não demonstra o design responsivo; no entanto; esses estilos servem apenas para gerar um resultado interessante):

```
css

.navbar {
    background-color: lightslategray;
    font-size: 1em;
```

```

        font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
        color: white;
        padding: 8px 5px 8px 5px;
    }

.navbar a {
    text-decoration: none;
    color: inherit;
}

.navbar-brand {
    font-size: 1.2em;
    font-weight: 600;
}

.navbar-item {
    font-variant: small-caps;
    margin-left: 30px;
}

.body-content {
    padding: 5px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

```

3. Modifique *templates/index.html* para se referir ao modelo base e substituir o bloco de conteúdo. Veja que, usando a herança, esse modelo se torna simples:

HTML

```

{% extends "layout.html" %}
{% block content %}
<span class="message">{{ message }}</span>{{ content }}
{% endblock %}

```

4. Modifique *templates/about.html* para se referir também ao modelo base e substituir o bloco de conteúdo:

HTML

```

{% extends "layout.html" %}
{% block content %}
{{ content }}
{% endblock %}

```

5. Execute o servidor para observar os resultados. Quando terminar, feche o navegador.

Hello, Flask! on Tuesday, 15 May, 2018 at 12:29:37

© 2018

6. Como você fez alterações significativas no aplicativo, é novamente um bom momento para [confirmar suas alterações no controle do código-fonte](#).

Próximas etapas

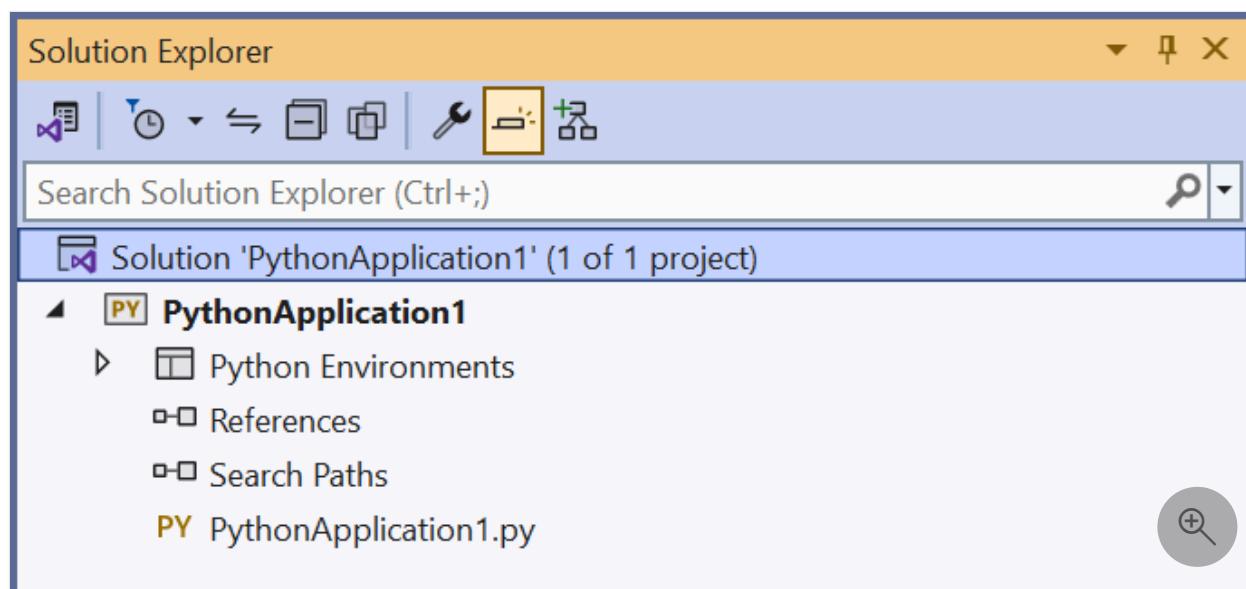
Você pode se aprofundar com estes recursos:

- Para ver mais funcionalidades de modelos Jinja, como o fluxo de controle, confira [Jinja Template Designer Documentation](#) (Documentação de Designer de Modelo do Jinja) (jinja.pocoo.org)
- Para obter detalhes sobre o uso de `url_for`, confira [url_for](#) na documentação do objeto do Aplicativo Flask (flask.pocoo.org)
- Código-fonte do tutorial no GitHub: [Microsoft/python-sample-vs-learning-flask](#)

Projetos do Python no Visual Studio

Artigo • 18/04/2024

Geralmente, os aplicativos Python são definidos usando apenas pastas e arquivos. Essa estrutura pode se tornar complexa à medida que os aplicativos crescem e acabam envolvendo arquivos gerados automaticamente, como JavaScript para aplicativos Web e assim por diante. Um [projeto do Visual Studio](#) pode ajudar você a gerenciar a complexidade. O projeto (um arquivo `.pyproj`) identifica todos os arquivos de origem e de conteúdo associados a ele. Ele também contém informações de build de cada arquivo, mantém as informações para integração com sistemas de controle de código-fonte e ajuda a organizar o aplicativo em componentes lógicos.



Os projetos são sempre gerenciados em uma [solução do Visual Studio](#). Uma solução pode conter vários projetos que podem fazer referência uns aos outros, como um projeto do Python que faz referência a um projeto do C++ que implementa um módulo de extensão. Com essa relação, o Visual Studio compila automaticamente o projeto em C++ (se for necessário) ao iniciar a depuração do projeto em Python. Para obter mais informações, consulte [Soluções e projetos no Visual Studio](#).

O Visual Studio fornece vários modelos de projeto do Python para criar rapidamente vários tipos de estruturas de aplicativos. Você pode escolher um modelo para criar um projeto em uma árvore de pastas existente ou criar um projeto limpo e vazio. Para obter uma lista de modelos disponíveis, consulte a tabela na seção [Modelos de projeto](#).

Dicas para trabalhar com projetos do Python

Você não precisa usar projetos para executar código do Python no Visual Studio, mas há benefícios em fazer isso. Para começar, examine as considerações a seguir sobre como

trabalhar com projetos e o Python.

- No Visual Studio 2019 e posterior, você pode abrir uma pasta com código do Python e executar o código sem criar arquivos de projeto e solução do Visual Studio.

As etapas guiadas para essa abordagem estão disponíveis no artigo [Início Rápido: Abrir e executar código do Python em uma pasta](#).

- Você não precisa de um projeto para executar código do Python no Visual Studio. Todas as versões do Visual Studio funcionam bem com códigos do Python.

Você pode abrir um arquivo do Python por conta própria e acessar imediatamente os recursos de preenchimento automático, IntelliSense e depuração. No entanto, há algumas possíveis desvantagens em trabalhar com o código sem um projeto:

- Como o código sempre usa o ambiente global padrão, talvez você veja preenchimentos incorretos ou erros, caso o código se destine a outro ambiente.
- O Visual Studio analisa todos os arquivos e pacotes na pasta da qual o arquivo é aberto. Esse processo pode consumir um tempo considerável de CPU.

- Você pode criar um projeto do Visual Studio com base no código existente. Essa abordagem é descrita na seção [Criar um projeto com arquivos existentes](#).

Tarefas básicas do projeto: arquivos, ambientes e inicialização

Ao usar projetos com seu código do Python, você realiza tarefas básicas, incluindo adicionar arquivos, atribuir um arquivo de inicialização e definir o ambiente do interpretador do Python.

À medida que desenvolve seu aplicativo, normalmente você precisa adicionar novos arquivos de diferentes tipos ao projeto. É fácil adicionar arquivos. Clique com o botão direito do mouse no projeto, selecione **Adicionar>Item existente** e navegue até encontrar o tipo de arquivo a ser adicionado. A opção **Adicionar>Novo Item** abre uma caixa de diálogo que mostra modelos de item que você pode usar para criar o arquivo. As opções incluem arquivos do Python vazios, uma classe do Python, um teste de unidade e vários arquivos relacionados a aplicativos Web. Explore as opções de modelos com um projeto de teste para saber o que está disponível em sua versão do Visual Studio. Para obter mais informações, veja a referência de [modelos de item](#).

Cada projeto do Python tem um arquivo de inicialização atribuído, que é mostrado em negrito no **Gerenciador de Soluções**. O arquivo de inicialização é executado quando

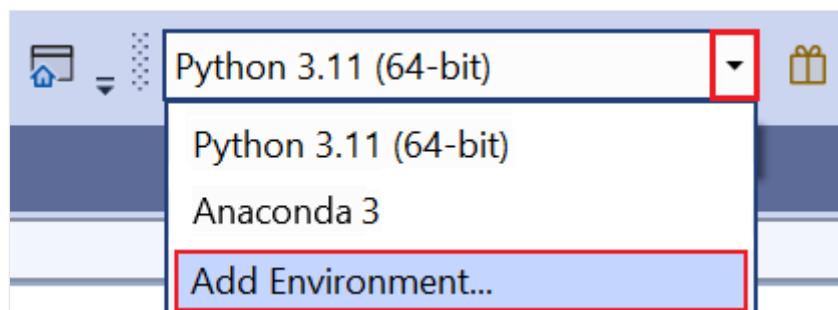
você começa a depuração (selecionando F5 ou **Depurar>Iniciar Depuração**) ou quando você executa o projeto na Janela **interativa**. Você pode abrir essa janela com o atalho de teclado Shift + Alt + F5 ou selecionando **Depurar>Executar Projeto no Python Interativo**. Para alterar o arquivo de inicialização, clique com o botão direito do mouse no arquivo que deseja usar e selecione **Definir como Item de Inicialização** (ou **Definir como Arquivo de Inicialização** em versões mais antigas do Visual Studio).

Se você remover o arquivo de inicialização selecionado de um projeto e não selecionar um arquivo alternativo, o Visual Studio não saberá com qual arquivo do Python usar para iniciar o projeto. Nesse caso, o Visual Studio 2017 versão 15.6 e posteriores mostrará um erro. Versões anteriores abrem uma janela de saída com o interpretador do Python em execução ou a janela de saída é aberta e, em seguida, fechada imediatamente. Se você observar algum desses comportamentos, verifique se haverá um arquivo de inicialização atribuído.

💡 Dica

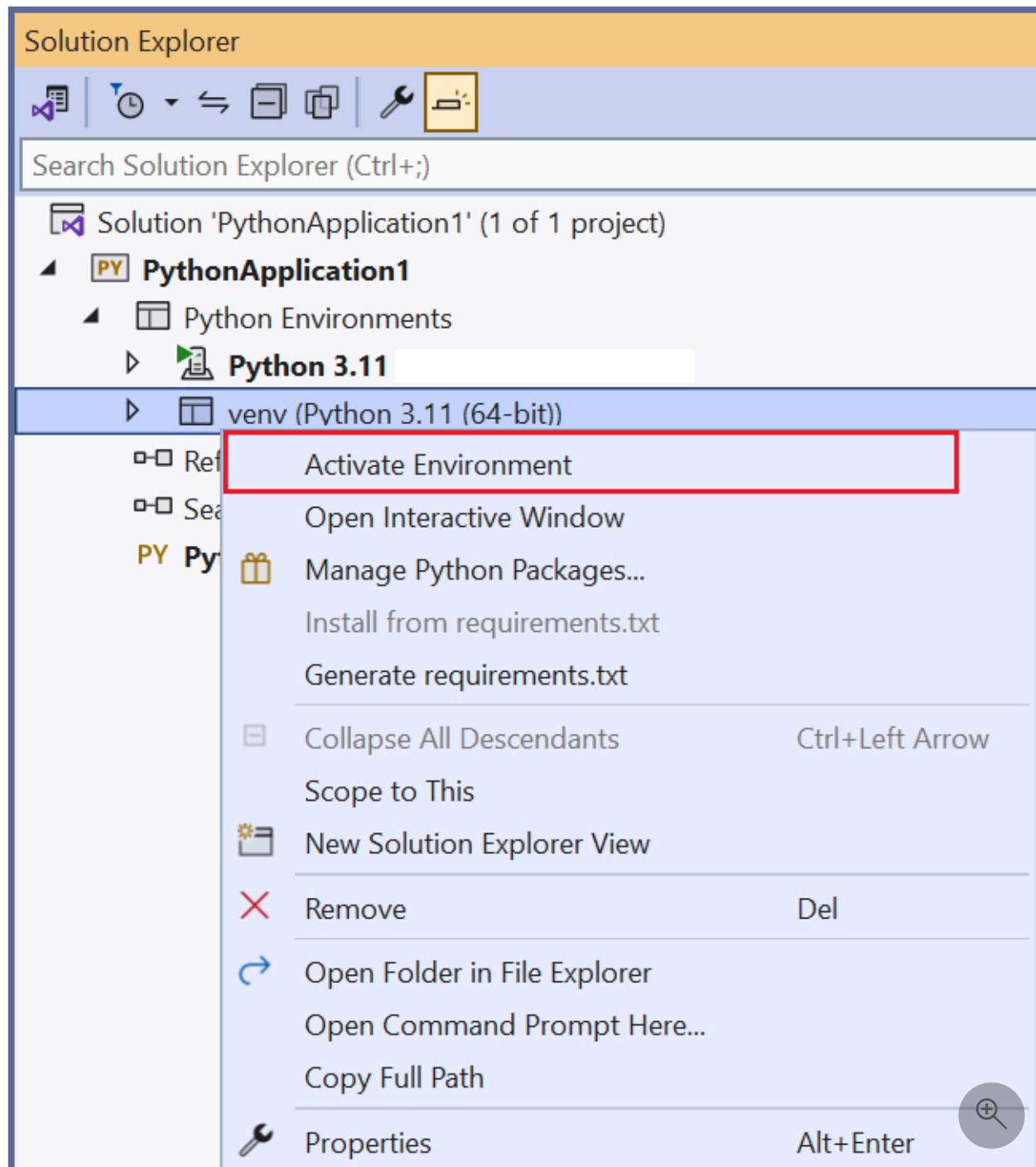
Para manter a janela de saída aberta, clique com o botão direito do mouse em seu projeto e selecione **Propriedades**. Na caixa de diálogo, selecione a guia **Depurar** e adicione o sinalizador `-i` ao campo **Argumentos do Interpretador**. Esse argumento faz com que o interpretador entre no modo interativo após a conclusão de um programa. A janela permanece aberta até você fechá-la, como usando o atalho de teclado **Ctrl+E+Enter**.

Um novo projeto sempre é associado ao ambiente global padrão do Python. Para associar o projeto a outro ambiente (incluindo ambientes virtuais), clique com o botão direito do mouse no nó **Ambientes do Python** no projeto. Selecione **Adicionar Ambiente** e selecione os ambientes desejados. Você também pode usar o controle de lista suspensa de ambientes na barra de ferramentas para selecionar outro ambiente para o projeto.



Para alterar o ambiente ativo, clique com o botão direito do mouse no ambiente desejado no **Gerenciador de Soluções** e selecione **Ativar Ambiente**, conforme mostrado

na imagem a seguir. Para obter mais informações, confira [Selecionar um ambiente para um projeto](#).



Modelos de projeto

O Visual Studio fornece muitas maneiras para configurar um projeto do Python, do zero ou com um código existente. Para usar um modelo, selecione **Arquivo > Novo > Projeto** ou clique com o botão direito do mouse na solução no **Gerenciador de Soluções** e selecione **Adicionar > Novo Projeto**. Na caixa de diálogo **novo projeto**, você pode ver modelos específicos do Python pesquisando em *Python* ou selecionando o nó **Linguagem > Python**:

Create a new project

[Clear all](#)

| Recent project templates | All languages |
|------------------------------------|---------------|
| Python Application | Python |
| Console App | Visual Basic |
| Console App (.NET Framework) | Visual Basic |
| Console App (.NET Framework) | C# |
| ASP.NET Core Web App (Razor Pages) | C# |
| WPF Application | C# |
| Empty Project (.NET Framework) | Visual Basic |
| Blank Solution | |

Python Application
A project for creating a command-line application

From Existing **Python** code
Create a new project using code files that are already in a folder hierarchy

Python Extension Module
Implements a native extension module for CPython 3.6 or later

Web Project
A project for creating a generic **Python** web project

Django Web Project
A project for creating an application using the Django web framework. It features sample pages that use the Twitter Bootstrap framework for responsive web design.

Next

Os modelos a seguir estão disponíveis no Visual Studio versão 2022.

[+] Expandir a tabela

| Modelo | Descrição |
|---|---|
| Com base em um código existente do Python | Cria um projeto do Visual Studio com base em um código existente do Python em uma estrutura de pastas. |
| Aplicativo do Python | Fornece uma estrutura de projeto básica para um novo aplicativo do Python com um único arquivo de origem vazio. Por padrão, o projeto é executado no interpretador do console do ambiente global padrão. Você pode alterar atribuindo um ambiente diferente . |
| Projetos Web | Projetos para aplicativos Web baseados em várias estruturas, incluindo Bottle, Django e Flask. |
| Aplicativo em segundo plano (IoT) | Dá suporte à implantação de projetos do Python a serem executados como serviços em segundo plano em dispositivos. Para obter mais informações, confira a Central de desenvolvedores de IoT do Windows . |
| Módulo de Extensão do Python | Esse modelo será exibido no Visual C++ se você instalar as ferramentas de desenvolvimento nativo do Python com a carga de trabalho do Python no Visual Studio 2017 ou posteriores (consulte Instalação). O modelo fornece a estrutura principal para uma DLL de extensão do C++, semelhante à estrutura descrita em Criar uma extensão do C++ para Python . |

(!) Observação

Como o Python é uma linguagem interpretada, os projetos em Python no Visual Studio não produzem um executável autônomo como outros projetos de linguagem compilada, como C#. Para saber mais, confira [Perguntas e respostas](#).

Criar um projeto com base em um código existente

Siga estas etapas para criar um projeto com arquivos existentes.

(i) Importante

O processo a seguir não move nem copia nenhum arquivo de origem original. Se você quiser trabalhar com uma cópia de seus arquivos, primeiro duplique a pasta e, em seguida, crie o projeto.

1. Inicie o Visual Studio e selecione **Arquivo > Novo > Projeto**.
2. Na caixa de diálogo **Criar um projeto**, pesquise *python*, selecione o modelo **De código do Python existente** e **Avançar**.
3. Na caixa de diálogo **Configurar novo projeto**, insira um **Nome** e **Local** para o projeto, escolha a solução para contê-lo e selecione **Criar**.
4. No assistente **Criar novo projeto do código Python existente**, defina o **Caminho da pasta** para o código existente, defina um **Filtro** para tipos de arquivo e especifique os **Caminhos de pesquisa** necessários para o projeto e selecione **Avançar**. Se você não souber os caminhos de pesquisa, deixe o campo em branco.



Welcome to the Create New Project from Existing Python Code Wizard

Enter or browse to the folder containing your Python code.

We won't move any files from where they are now.

c:\proj\pyApp

...

Enter the filter for files to include.

Files with the .py extension are always included.

.pyw;.txt;*.htm;*.html;*.css;*.djt;*.js;*.ini;*.png;*.jpg;*.gif;*.bmp;*.ico;*.sv

Enter any search paths your project needs.

One on each line, and we'll make them relative to the project file for you.

c:\proj\pyApps\common

...

Back

Next

Finish



5. Na próxima página, selecione o **Arquivo de inicialização** do projeto. O Visual Studio seleciona o interpretador e a versão globais padrão do Python. Você pode alterar o ambiente usando o menu suspenso. Quando estiver pronto, selecione **Próximo**.

ⓘ Observação

A caixa de diálogo mostra apenas arquivos na pasta raiz. Se o arquivo desejado estiver em uma subpasta, deixe o arquivo de inicialização em branco. Você pode definir o arquivo de inicialização no **Gerenciador de Soluções**, conforme descrito em uma etapa posterior.



Welcome to the Create New Project from Existing Python Code Wizard

Select the Python interpreter and version to use.

This setting can be changed later in Project Properties.

(Global default or an auto-detected virtual environment)

Choose which file to run when F5 is pressed.

If it is not in this list, you can right-click any file in your project and choose "Set as startup file"

module1.py

test1.py

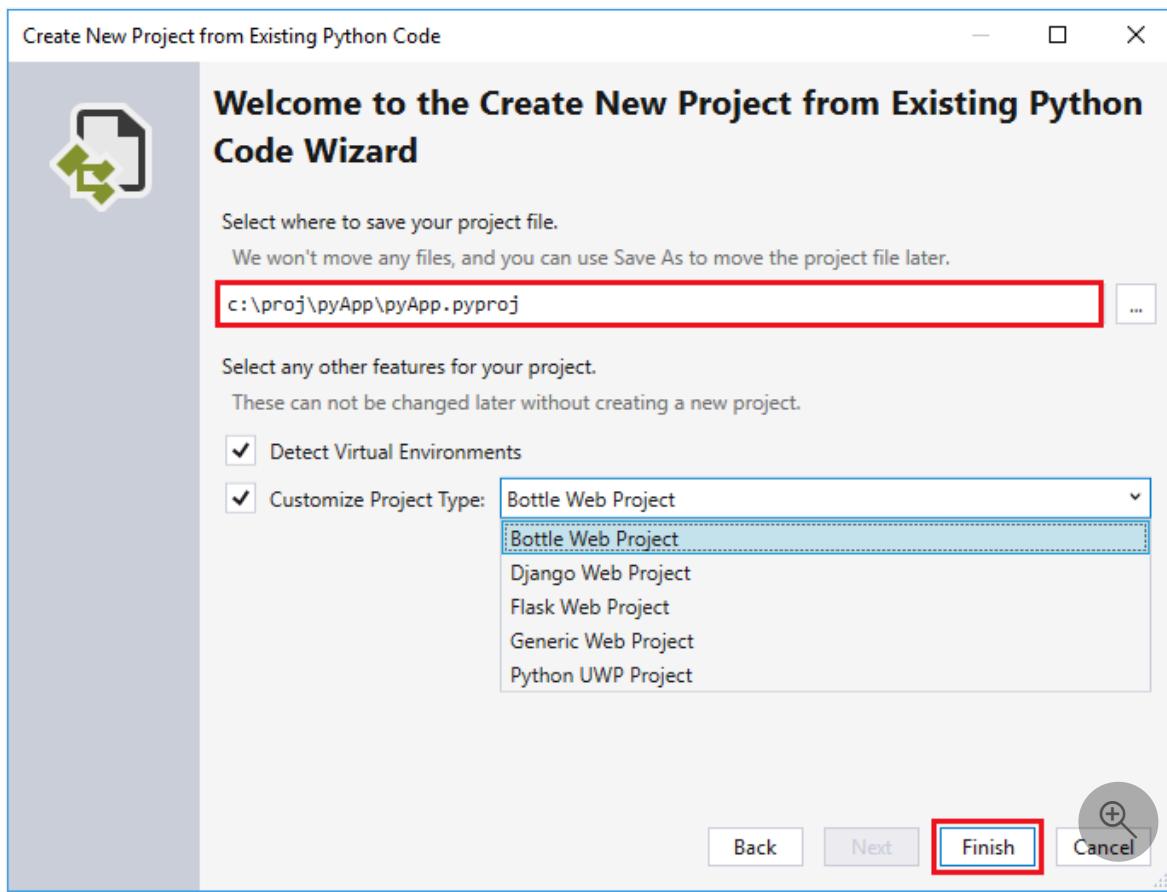
Back

Next

Finish



6. Selecione o local para armazenar o arquivo de projeto (um arquivo .pyproj no disco). Caso se aplique, também será possível incluir a detecção automática de ambientes virtuais e personalizar o projeto para outras estruturas da Web. Se você não tiver certeza sobre essas opções, deixe os campos com as configurações padrão.



7. Selecione Concluir.

O Visual Studio cria e abre o projeto no **Gerenciador de Soluções**. Se você quiser mover o arquivo `.pyproj` para um local diferente, selecione o arquivo no **Gerenciador de Soluções** e selecione **Arquivo > Salvar como** na barra de ferramentas. Essa ação atualizará as referências de arquivo no projeto, mas não moverá nenhum arquivo de código.

8. Para definir um arquivo de inicialização diferente, localize o arquivo no **Gerenciador de Soluções**, clique com o botão direito do mouse e selecione **Definir como Arquivo de Inicialização**.

Arquivos vinculados

Os arquivos vinculados são aqueles que são inseridos em um projeto, mas que geralmente residem fora das pastas do projeto do aplicativo. Esses arquivos aparecem no **Gerenciador de Soluções** como arquivos normais com um ícone de atalho sobreposto: 

Os arquivos vinculados são especificados no arquivo `.pyproj` usando o elemento `<Compile Include="...">`. Os arquivos vinculados serão **implícitos** se usarem um caminho relativo fora da estrutura de diretório. Se os arquivos usarem caminhos no

Gerenciador de Soluções, os arquivos vinculados serão **explícitos**. O exemplo a seguir mostra arquivos explicitamente vinculados:

XML

```
<Compile Include="..\test2.py">
    <Link>MyProject\test2.py</Link>
</Compile>
```

Arquivos vinculados são ignorados nas seguintes condições:

- O arquivo vinculado contém metadados do `Link` e o caminho especificado no atributo `Include` reside no diretório do projeto.
- O arquivo vinculado duplica um arquivo que existe na hierarquia do projeto.
- O arquivo vinculado contém metadados do `Link` e o caminho do `Link` é um caminho relativo fora da hierarquia do projeto.
- O caminho do link tem raiz.

Trabalhar com arquivos vinculados

Para adicionar um item existente como um link, clique com o botão direito do mouse na pasta do projeto em que deseja adicionar o arquivo e, em seguida, selecione **Adicionar>Item Existente**. Na caixa de diálogo, selecione um arquivo e, em seguida, selecione **Adicionar>Adicionar como Link**. Se não existirem arquivos conflitantes, esse comando criará um link na pasta selecionada. No entanto, o link não será adicionado se existir um arquivo com o mesmo nome ou se já existir um link para esse arquivo no projeto.

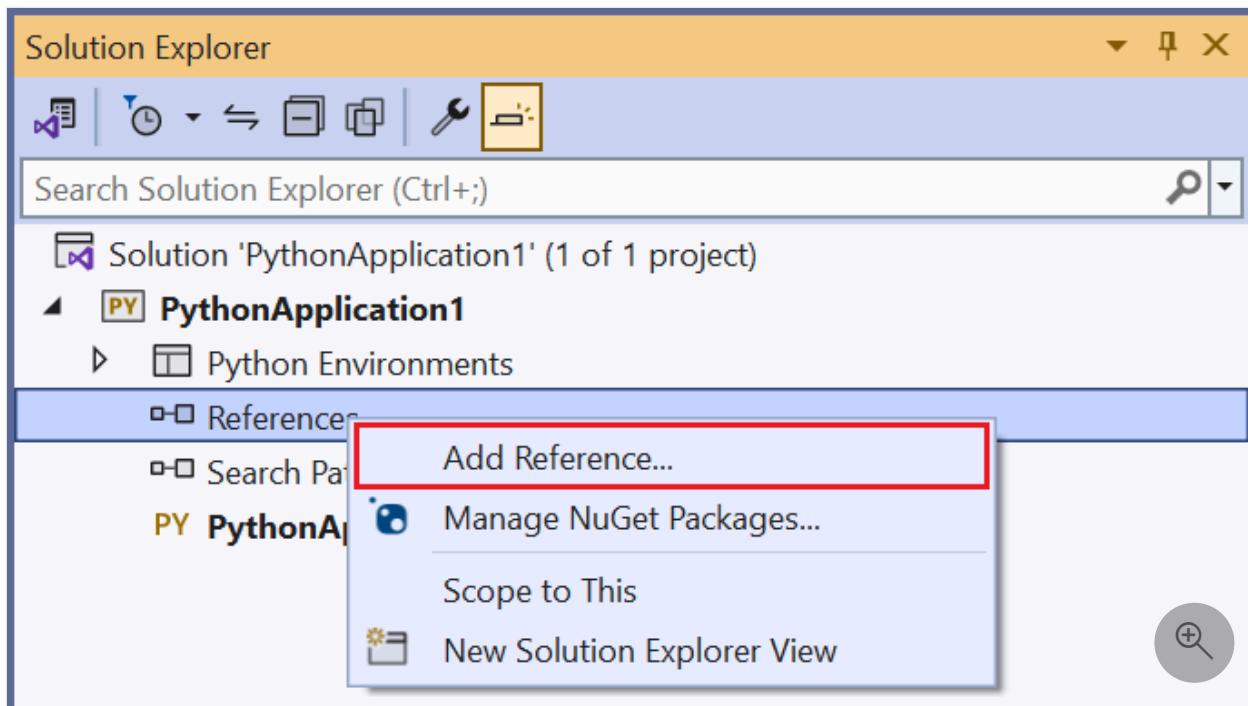
Se você tentar vincular a um arquivo que já existe nas pastas do projeto, ele será adicionado como um arquivo normal e não como um link. Para converter um arquivo em um link, selecione **Arquivo>Salvar como** para salvar o arquivo em um local fora da hierarquia do projeto. O Visual Studio converte automaticamente o arquivo em um link. Da mesma forma, um link pode ser convertido novamente usando a opção **Arquivo>Salvar Como** para salvar o arquivo em algum lugar na hierarquia do projeto.

Se você mover um arquivo vinculado no **Gerenciador de Soluções**, o link será movido, mas o arquivo real não será afetado. Da mesma forma, a exclusão de um link removerá o link sem afetar o arquivo.

Arquivos vinculados não podem ser renomeados.

Referências

Os projetos do Visual Studio dão suporte à adição de referências a projetos e extensões, que são exibidas no nó **Referências** do **Gerenciador de Soluções**:



Geralmente, referências de extensão indicam dependências entre projetos e são usadas para fornecer o IntelliSense em tempo de design ou a vinculação em tempo de compilação. Os projetos do Python usam referências de maneira semelhante, mas devido à natureza dinâmica do Python, elas são usadas principalmente em tempo de design para fornecer um IntelliSense avançado. Elas também podem ser usadas para implantação no Microsoft Azure para instalar outras dependências.

Trabalhar com módulos de extensão

Uma referência a um arquivo `.pyd` habilita o IntelliSense no módulo gerado. O Visual Studio carrega o arquivo `.pyd` no interpretador do Python e examina seus tipos e suas funções. O Visual Studio também tenta analisar as cadeias de caracteres doc em funções para fornecer ajuda da assinatura.

Se, a qualquer momento, o módulo de extensão é atualizado em disco, o Visual Studio analisa o módulo novamente em segundo plano. Essa ação não tem nenhum efeito no comportamento do runtime, mas alguns preenchimentos não estarão disponíveis até que a análise seja concluída.

Você também pode ter que adicionar um [caminho de pesquisa](#) à pasta que contém o módulo.

Trabalhar com projetos do .NET

Ao trabalhar com o IronPython, é possível adicionar referências aos assemblies do .NET para habilitar o IntelliSense. Em projetos do .NET, clique com o botão direito do mouse no nó **Referências** em seu projeto do Python e selecione **Adicionar Referência**. Na caixa de diálogo, selecione a guia **Projetos** e navegue até o projeto desejado. Para as DLLs baixadas separadamente, selecione a guia **Procurar** e procure a DLL desejada.

Como as referências no IronPython não estão disponíveis até depois de uma chamada ao método `clr.AddReference('<AssemblyName>')`, você também precisa adicionar uma chamada de método `clr.AddReference` apropriada ao assembly. Essa chamada normalmente é adicionada no início do código. Por exemplo, o código criado pelo modelo de projeto **Aplicativo Windows Forms do IronPython** (disponível no Visual Studio 2019) do Visual Studio inclui duas chamadas no início do arquivo:

```
Python

import clr
clr.AddReference('System.Drawing')
clr.AddReference('System.Windows.Forms')

from System.Drawing import *
from System.Windows.Forms import *

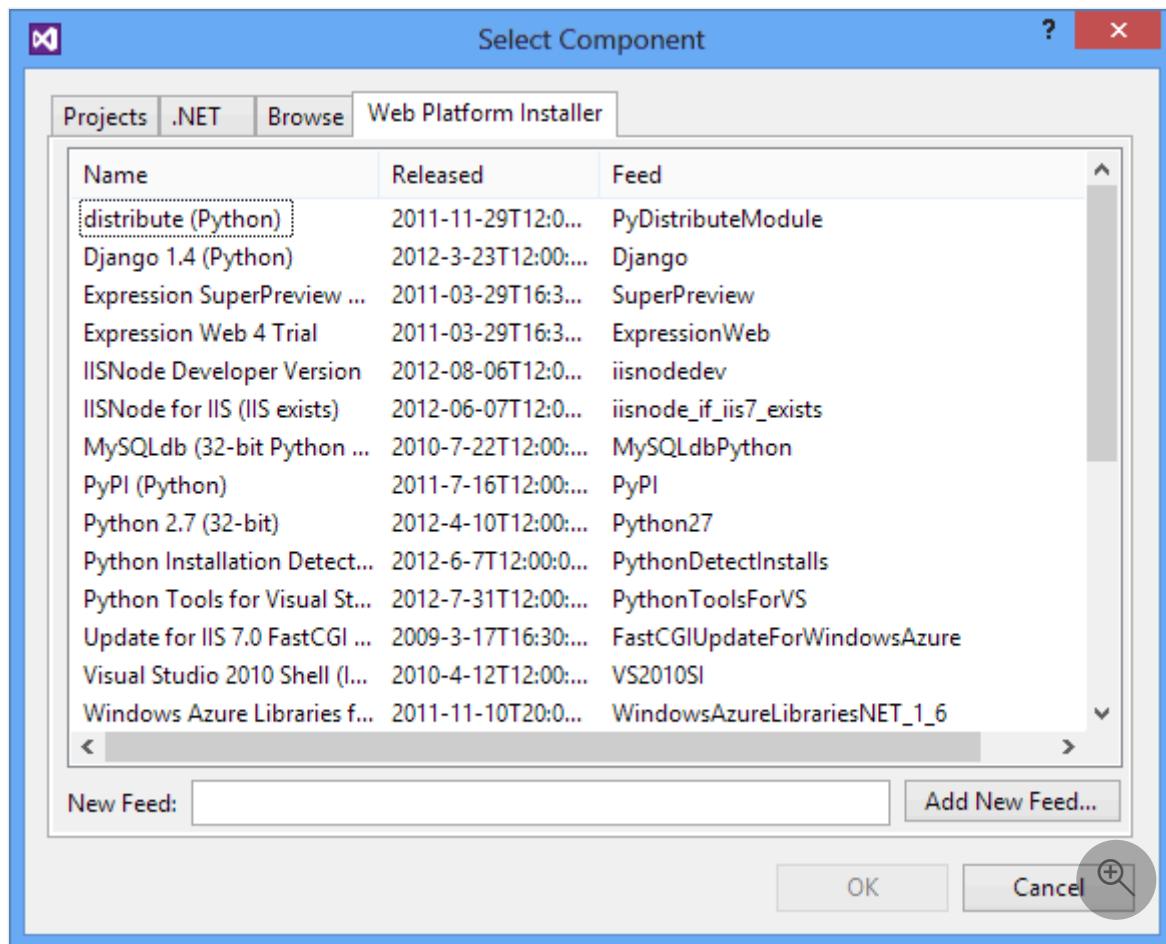
# Other code omitted
```

Trabalhar com projetos do WebPI

É possível adicionar referências a entradas de produtos do WebPI (Web Platform Installer) para implantação nos Serviços de Nuvem do Microsoft Azure, em que é possível instalar componentes adicionais por meio do feed do WebPI. Por padrão, o feed exibido é específico ao Python e inclui o Django, o CPython e outros componentes básicos. Você também pode selecionar seu feed, conforme mostrado na imagem a seguir. Ao publicar no Microsoft Azure, uma tarefa de instalação instala todos os produtos referenciados.

ⓘ Importante

Os projetos do WebPI não estão disponíveis no Visual Studio 2017 nem no Visual Studio 2019.



Conteúdo relacionado

- Soluções e projetos no Visual Studio
- Selecionar um ambiente para um projeto
- Instalar o suporte de Python no Visual Studio

Comentários

Esta página foi útil?

Yes

No

Modelos de projeto de aplicativo Web Python

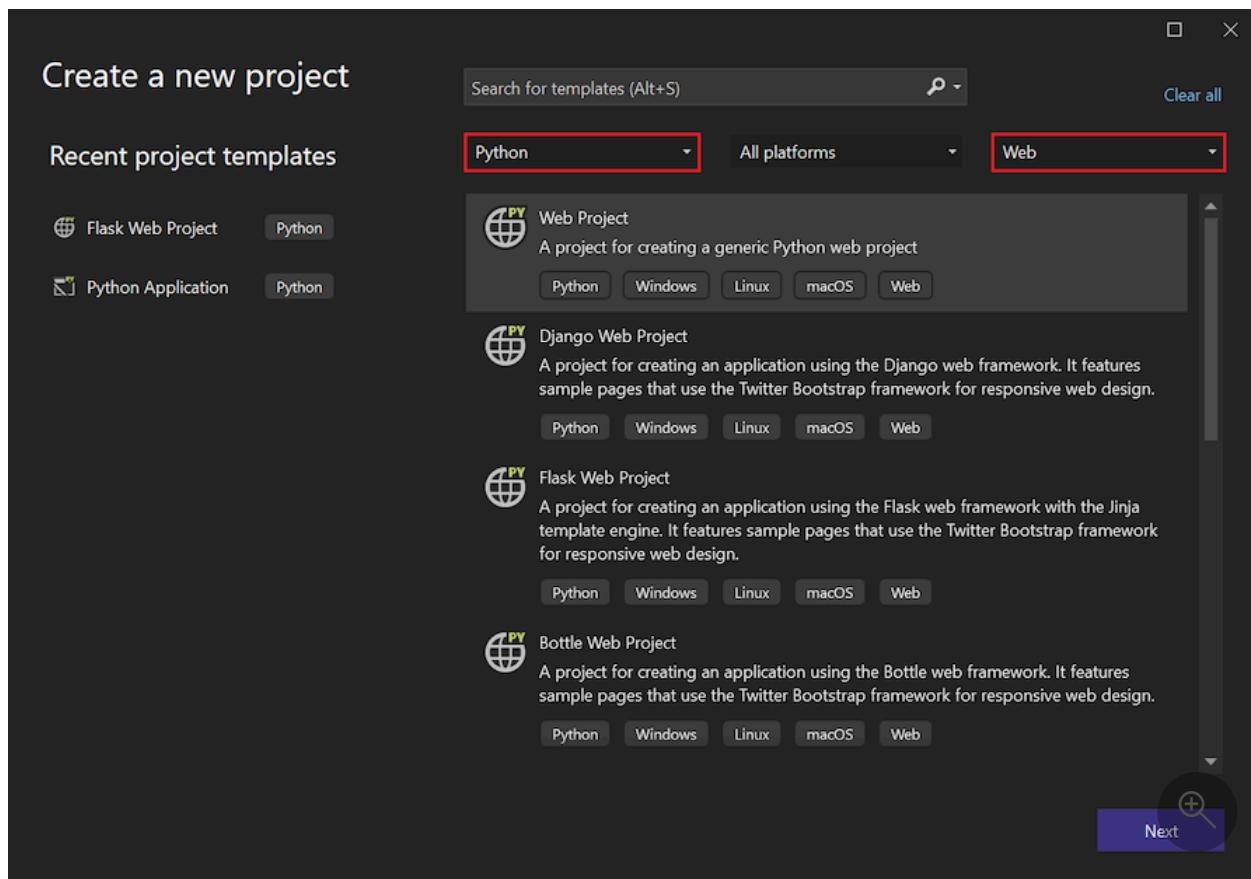
Artigo • 18/04/2024

O Python no Visual Studio é compatível com o desenvolvimento de projetos da Web nas estruturas Bottle, Flask e Django por meio de modelos de projeto e um inicializador de depuração que pode ser configurado para manipular várias estruturas. Esses modelos incluem um arquivo *requirements.txt* para declarar as dependências necessárias. Ao criar um projeto de um desses modelos, o Visual Studio solicita que você instale os pacotes dependentes, conforme descrito em [Requisitos de instalação](#) posteriormente neste artigo.

Você também pode usar o modelo genérico **Projeto Web** para outras estruturas, como Pyramid. Nesse caso, nenhuma estrutura é instalada com o modelo. Em vez disso, você instala os pacotes necessários no ambiente que está usando para o projeto. Para obter mais informações, consulte [Janela de ambientes do Python – guia Pacote](#).

Opções de modelo de projeto

Crie um projeto a partir de um modelo selecionando **Arquivo>Novo>Projeto** no menu da barra de ferramentas. Na caixa de diálogo **Criar um novo projeto**, você pode filtrar a lista de modelos para ver as opções disponíveis para projetos Web em Python. Insira os termos-chave na caixa **Pesquisar** ou use os menus suspensos de filtro para selecionar **Python** como a linguagem e **Web** como o tipo de projeto.



Depois de selecionar um modelo, forneça um nome para o projeto e para a solução e defina opções para o diretório da solução e para o repositório Git.

O modelo genérico de **Projeto Web** fornece apenas um projeto vazio do Visual Studio sem nenhum código e nenhuma suposição diferente além de ser um projeto do Python. Os outros modelos se baseiam nas estruturas da Web Bottle, Flask ou Django e são agrupados em três categorias, conforme descrito nas seções a seguir. Os aplicativos criados por um desses modelos contêm código suficiente para executar e depurar o aplicativo localmente. Cada modelo também fornece o [objeto de aplicativo WSGI](#) necessário (python.org) para uso com servidores da Web de produção.

Grupo em branco

Todos os modelos de Projeto Web **Blank <framework>** criam um projeto com código boilerplate mais ou menos mínimo e as dependências necessárias declaradas em um arquivo *requirements.txt*.

[\[+\] Expandir a tabela](#)

| Modelo | Descrição |
|---------------------------------|---|
| Projeto Web em Branco do Bottle | Gera um aplicativo mínimo no arquivo <i>app.py</i> com uma home page do local / e uma página /hello/<name> que ecoa o valor <name> usando um modelo de página embutido muito curto. |

| Modelo | Descrição |
|---------------------------------|--|
| Projeto Web em Branco do Django | Gera um projeto Django com a estrutura do site principal do Django, mas não aplicativos Django. Para obter mais informações, confira Modelos do Django e Etapa 1 do tutorial – Conheça o Django . |
| Projeto Web em Branco do Flask | Gera um aplicativo mínimo com um único "Olá, Mundo!" para o local /. Este aplicativo é semelhante ao resultado das seguintes etapas detalhadas em Início rápido: use o Visual Studio para criar seu primeiro aplicativo Web Python . Para obter mais informações, consulte Etapa 1 do tutorial – Conheça o Flask . |

Grupo da Web

Todos os modelos de Projeto Web <Framework> criam um aplicativo Web inicial com um design idêntico, seja qual for a estrutura escolhida. O aplicativo tem as páginas Início, Sobre e Contato, juntamente com uma barra de menus de navegação e um design que usa a Inicialização. Cada aplicativo é configurado adequadamente para fornecer arquivos estáticos (CSS, JavaScript e fontes) e usa um mecanismo de modelo de página adequado para a estrutura.

 [Expandir a tabela](#)

| Modelo | Descrição |
|-----------------------|--|
| Projeto Web do Bottle | Gera um aplicativo cujos arquivos estáticos estão contidos na pasta <i>static</i> e são manipulados por meio do código no arquivo <i>app.py</i> . O roteamento para as páginas individuais está contido no arquivo <i>routes.py</i> . A pasta <i>views</i> contém os modelos de página. |
| Projeto Web do Django | Gera um projeto e um aplicativo do Django com três páginas, suporte de autenticação e um banco de dados SQLite (mas nenhum modelo de dados). Para obter mais informações, confira Modelos do Django e Etapa 4 do tutorial – Conheça o Django . |
| Projeto Web do Flask | Gera um aplicativo cujos arquivos estáticos estão contidos na pasta <i>static</i> . O código no arquivo <i>views.py</i> manipula o roteamento, com modelos de página que usam o mecanismo Jinja contido na pasta <i>templates</i> . O arquivo <i>runserver.py</i> fornece o código de inicialização. |

Requisitos de instalação

Ao criar um projeto com base em um modelo específico à estrutura, o Visual Studio exibe uma caixa de diálogo para ajudar você a instalar os pacotes necessários usando o PIP. Também recomendamos o uso de um [ambiente virtual](#) para projetos Web para garantir que as dependências corretas sejam incluídas durante a publicação do site:

This project requires external packages

We can download and install these packages for you automatically, but we need to know whether you want to install them for just this project or for all your projects.

→ [Install into a virtual environment...](#)

Packages will only be available for this project (recommended)

→ [Install into Python 3.6 \(64-bit\)](#)

Packages will be shared by all projects

→ [I will install them myself](#)

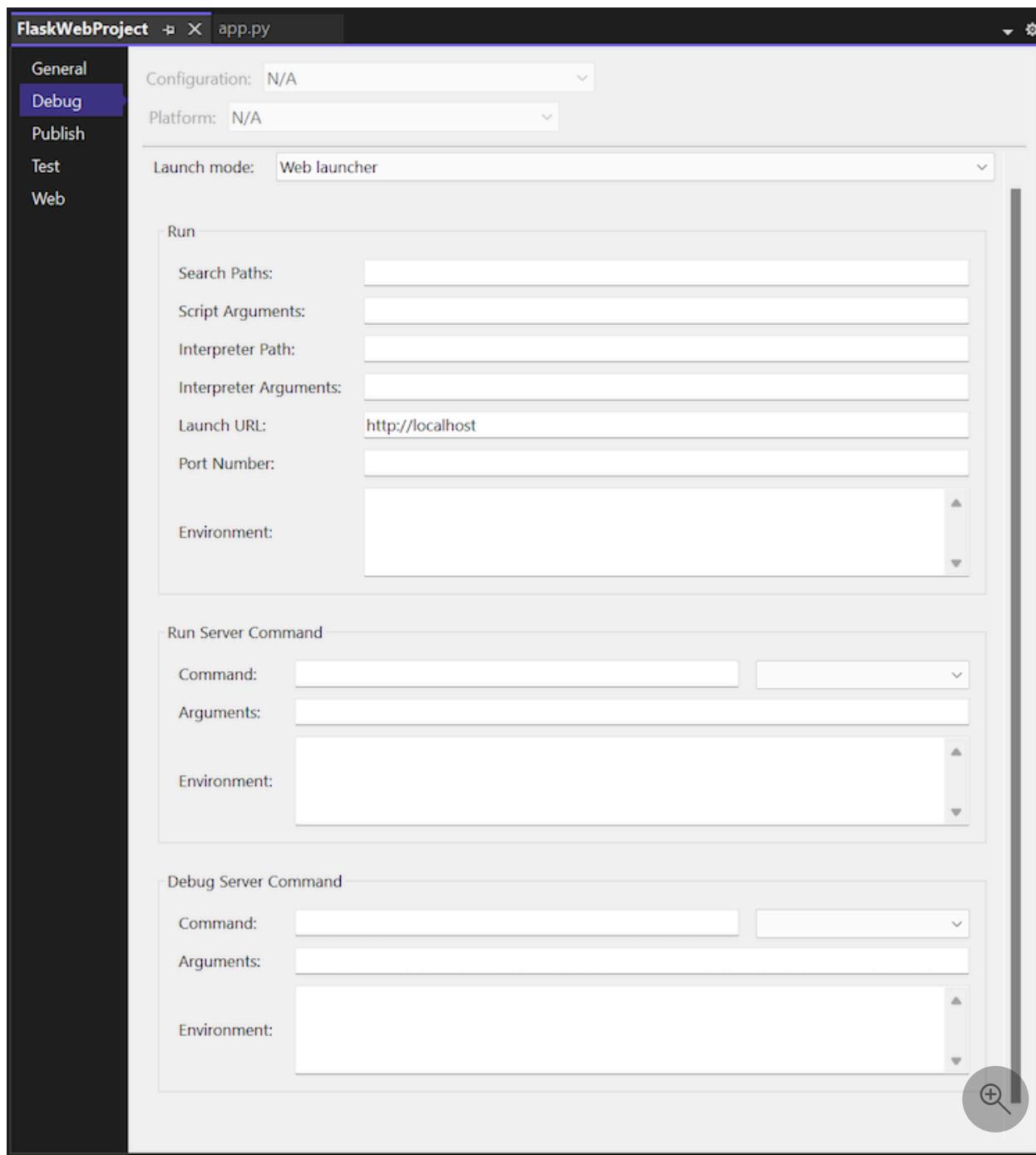
 [Show required packages](#)

Se for usar o controle do código-fonte, normalmente, você omitirá a pasta de ambiente virtual, pois esse ambiente poderá ser recriado usando somente o arquivo *requirements.txt*. A melhor maneira de excluir a pasta é primeiro selecionar a opção **Eu os instalarei sozinho** e desabilitar a confirmação automática antes de criar o ambiente virtual. Para obter mais informações, confira [Examinar os controles do Git](#) no [Tutorial Aprenda Django](#) e no [Tutorial Aprenda Flask](#).

Ao implantar o Serviço de Aplicativo do Microsoft Azure, selecione uma versão do Python como uma [extensão de site](#) e instalar os pacotes manualmente. Além disso, como o Serviço de Aplicativo do Azure **não** instala pacotes automaticamente de um arquivo *requirements.txt* quando implantado por meio do Visual Studio, siga os detalhes de configuração em aka.ms/PythonOnAppService.

Opções de depuração

Ao abrir um projeto Web para depuração, o Visual Studio inicia um servidor Web local em uma porta aleatória e abre seu navegador padrão para esse endereço e porta. Para especificar mais opções, clique com o botão direito do mouse no projeto no **Gerenciador de Soluções** e selecione **Propriedades**. Na página **Propriedades**, selecione a guia **Depurar**.



Há três grupos de opções de configuração comuns para depurar o projeto. O grupo **Executar** inclui as seguintes propriedades:

- As opções **Caminhos de Pesquisa**, **Argumentos de Script**, **Caminho do Interpretador** e **Argumentos do Interpretador** são as mesmas da [depuração normal](#).
- O **URL de Inicialização** especifica a URL que é aberta no navegador. A localização padrão é `localhost`.
- O **Número da Porta** identifica a porta a ser usada se nenhuma for especificada na URL (o Visual Studio seleciona uma automaticamente por padrão). Essa configuração permite substituir o valor padrão da variável de ambiente `SERVER_PORT`, que é usada pelos modelos para configurar qual porta o servidor de depuração local escuta.

- A lista **Ambiente** define variáveis a serem definidas no processo gerado. O formato é uma lista de pares `<NAME>=<VALUE>` separada por nova linha.

As propriedades nos grupos **Executar Comando do Servidor** e **Depurar Comando do Servidor** determinam como o servidor Web é iniciado. Como muitas estruturas exigem o uso de um script fora do projeto atual, o script pode ser configurado aqui e o nome do módulo de inicialização pode ser passado como um parâmetro.

- O **Command** pode ser um script do Python (arquivo `*.py`), um nome de módulo (como em `python.exe -m module_name`) ou uma única linha de código (como em `python.exe -c "code"`). O valor na caixa suspensa indica qual tipo é pretendido.
- A lista de **Argumentos** é passada na linha de comando após o comando.
- Novamente, a lista **Ambiente** define variáveis a serem estabelecidas após todas as propriedades e que podem modificar o ambiente, como o número da porta e os caminhos de pesquisa. Esses valores de variável podem substituir outros valores de propriedade.

Qualquer propriedade de projeto ou variável de ambiente pode ser especificada com a sintaxe do MSBuild, como `$(StartupFile) --port $(SERVER_PORT)`. `$(StartupFile)` é o caminho relativo para o arquivo de inicialização e `{StartupModule}` é o nome importável do arquivo de inicialização. `$(SERVER_HOST)` e `$(SERVER_PORT)` são variáveis de ambiente normais definidas pelas propriedades **URL de Inicialização** e **Número da Porta**, automaticamente ou pela propriedade **Ambiente**.

ⓘ Observação

Os valores no **Executar Comando do Servidor** são usados com o comando **Debug>Start Server** ou o atalho de teclado **Ctrl+F5**. Os valores no grupo **Depurar Comando do Servidor** são usados com o comando **Debug>Start Debug Server** ou **F5**.

Configuração do Bottle de exemplo

O modelo de **Projeto Web Bottle** inclui um código de texto clichê que faz a configuração necessária. Um aplicativo importado do Bottle pode não incluir esse código; no entanto, nesse caso, as seguintes configurações iniciam o aplicativo usando o módulo `bottle` instalado:

- Grupo **Executar Comando do Servidor**:
 - **Comando:** `bottle` (módulo)
 - **Argumentos:** `--bind=%SERVER_HOST%:%SERVER_PORT% {StartupModule}:app`

- Grupo Depurar Comando do Servidor:
 - Comando: `bottle` (módulo)
 - Argumentos `--debug --bind=%SERVER_HOST%:%SERVER_PORT% {StartupModule}:app`

A opção `--reload` não é recomendada ao usar o Visual Studio para depuração.

Configuração de exemplo do Pyramid

Atualmente, a melhor forma de criar aplicativos do Pyramid é usando a ferramenta de linha de comando `pcreate`. Depois de criar o aplicativo, ele poderá ser importado usando o modelo [Com base em um código existente do Python](#). Depois de terminar a importação, selecione a personalização [Projeto Web Genérico](#) para configurar as opções. Essas configurações presumem que o Pyramid está instalado em um ambiente virtual no local `..\env`.

- Grupo Run:
 - Número da Porta: 6543 (ou o que estiver configurado nos arquivos `.ini`)
- Grupo Executar Comando do Servidor:
 - Comando: `..\env\scripts\pserv-script.py` (script)
 - Argumentos: `Production.ini`
- Grupo Depurar Comando do Servidor:
 - Comando: `..\env\scripts\pserv-script.py` (script)
 - Argumentos: `Development.ini`

Dica

Provavelmente, será necessário configurar a propriedade **Diretório de Trabalho** do projeto, pois os aplicativos do Pyramid estão normalmente uma pasta abaixo da raiz do projeto.

Outras configurações

Se você tiver configurações para outra estrutura que gostaria de compartilhar ou se gostaria de solicitar configurações para outra estrutura, abra um [problema no GitHub](#).

Conteúdo relacionado

- [Referência de modelos de item do Python](#)

- Selecionar um ambiente Python para um projeto no Visual Studio
-

Comentários

Esta página foi útil?

 Yes

 No

Modelos de projeto Web do Django para o Python no Visual Studio

Artigo • 18/04/2024

O [Django](#) é uma estrutura do Python de alto nível projetada para um desenvolvimento da Web rápido, seguro e escalonável. O suporte do Python no Visual Studio fornece vários modelos de projeto para configurar a estrutura de um aplicativo Web baseado em Django.

Os modelos estão disponíveis no Visual Studio em Arquivo>Novo>Projeto e incluem o **Projeto Web do Django em Branco** e o **Projeto Web do Django**. Para obter um passo a passo dos modelos, confira a série de tutoriais [Tutorial: Introdução à estrutura da Web do Django no Visual Studio](#).

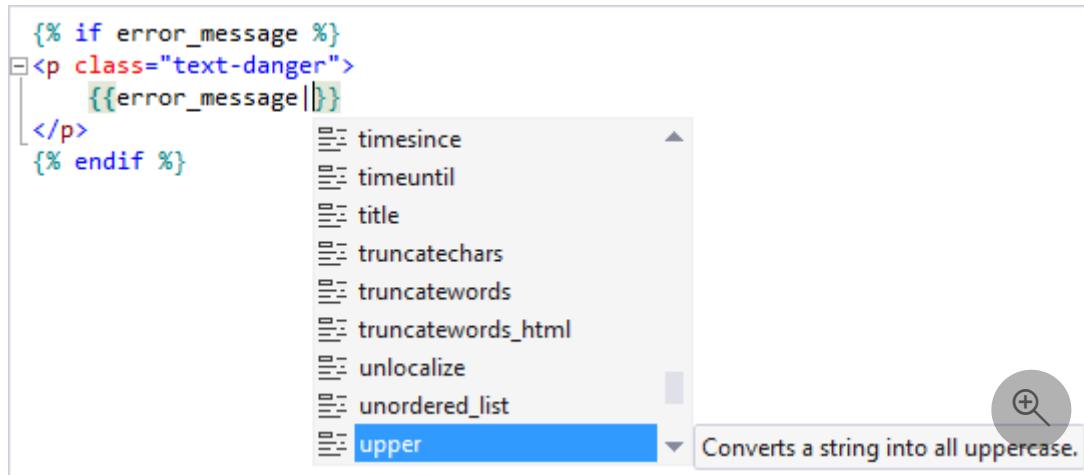
Recursos do IntelliSense

O Visual Studio fornece suporte completo do IntelliSense para projetos do Django, incluindo os seguintes recursos:

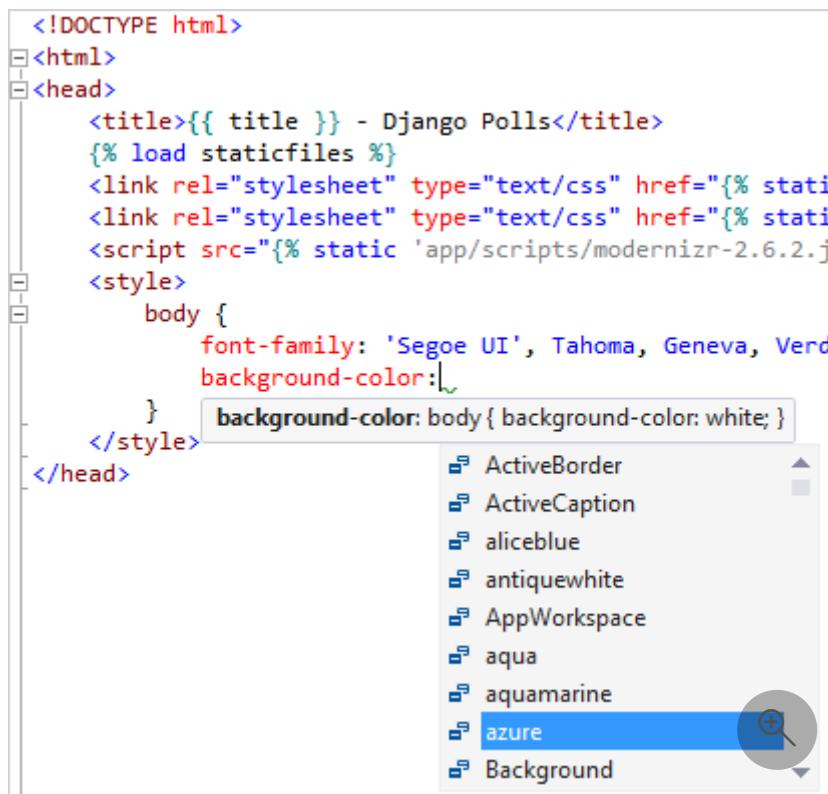
- Variáveis de contexto passadas para o modelo:



- Marcação e filtragem para itens internos e definidos pelo usuário:



- Realce de sintaxe para elementos incorporados de CSS e JavaScript:



The screenshot shows a code editor window with a portion of a Django template file. The template includes HTML, CSS links, and a JavaScript block. A tooltip is displayed over the word 'text' in the JavaScript code, listing various 'Text'-related DOM elements and global variables. The 'text' item is highlighted, and a note indicates it is a '(global variable)'. A search icon is visible in the bottom right corner of the tooltip.

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }} - Django Polls</title>
    {% load staticfiles %}
    <link rel="stylesheet" type="text/css" href="{% static 'app/content/b0
    <link rel="stylesheet" type="text/css" href="{% static 'app/content/si
    <script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>
    <script type="text/javascript" language="javascript">
        var text = 'hello';
        alert(tex
    </sc> alert([String message])
</head>
```

Suporte para depuração

O Visual Studio também fornece [suporte de depuração](#) completo para projetos do Django:

```

index.html ✘
8   {% if latest_poll_list %}
9     <table class="table table-hover">
10    <tbody>
11      {% for poll in latest_poll_list %}
12        <tr>
13          <td>
14            <a href="{% url 'app:detail' poll.id %}">{{poll.text}}</a>
15          </td>
16        </tr>
17      {% endfor %}
18    </tbody>
19  </table>

```

Watch 1

| Name | Type |
|---------------|--|
| poll.text | unicode |
| poll.id | int |
| poll.pub_date | datetime.datetime(2014, 8, 12, 21, 3, 7, 331000, tzinfo=<UTC>) |
| day | int |

Call Stack

| Name | Language |
|------------------------------|------------------|
| render in index line 14 | Django Templates |
| render in index line 11 | Django Templates |
| render_node in debug line 78 | Python |
| render in base line 840 | Python |
| render in index line 8 | Django Templates |

Call Stack Breakpoints Immediate Window Output

Console de gerenciamento do Django

O console de gerenciamento do Django é acessado por meio de vários comandos no menu **Projeto** ou clicando com o botão direito do mouse no projeto do Django em **Gerenciador de Soluções**.

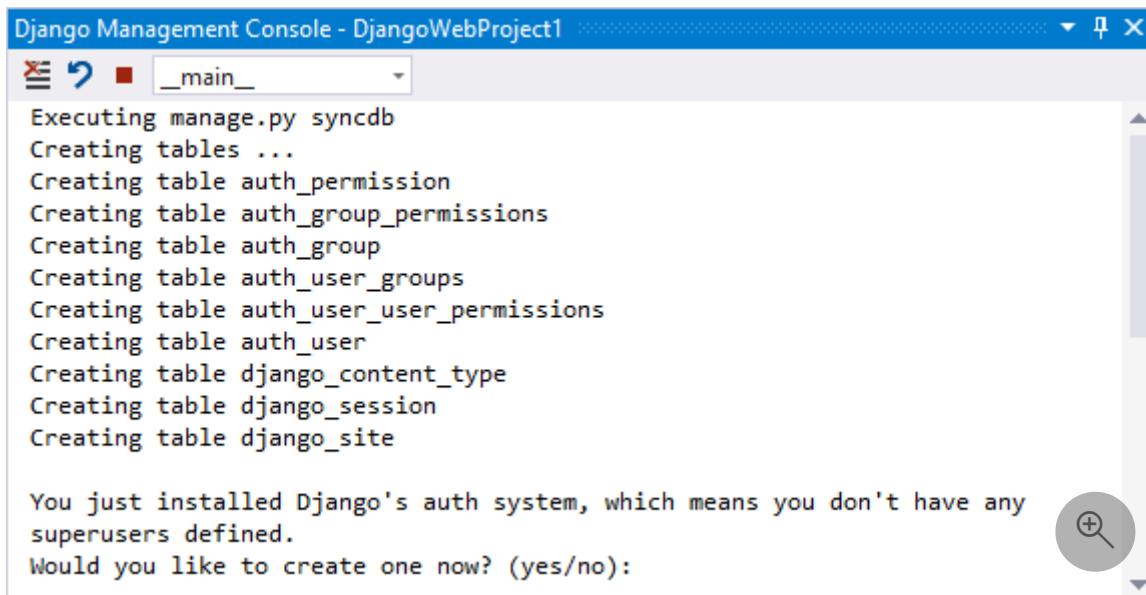
- **Open Django Shell:** abre um shell no contexto do aplicativo que permite manipular os modelos:

```

Django Management Console - DjangoWebProject1
>>> from app.models import Poll, Choice
>>> from django.utils import timezone
>>> p = Poll(text="Favorite way to host Django on Azure?", pub_date=timezone.now())
>>> p.save()
>>> p.choice_set.create(text='Cloud Service', votes=0)
<Choice: Cloud Service>
>>> p
<Poll: Favorite way to host Django on Azure?>
>>> |

```

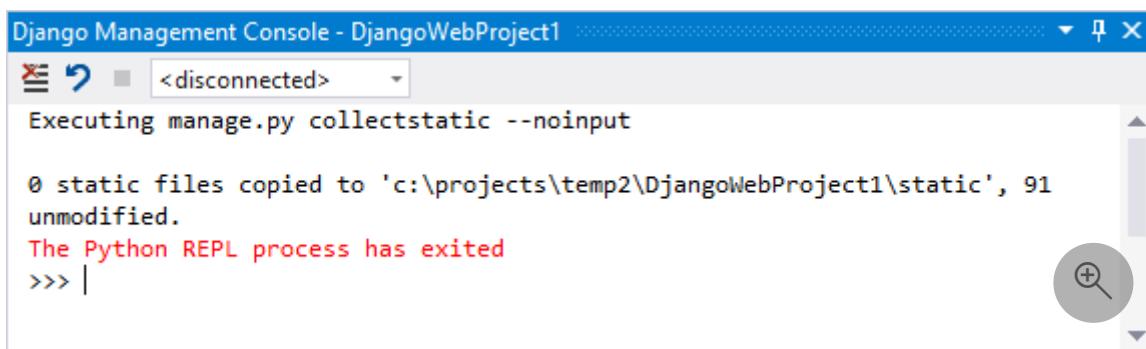
- **Django Sync DB:** executa o comando `manage.py syncdb` em uma Janela Interativa:



```
Django Management Console - DjangoWebProject1
__main__
Executing manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site

You just installed Django's auth system, which means you don't have any
superusers defined.
Would you like to create one now? (yes/no):
```

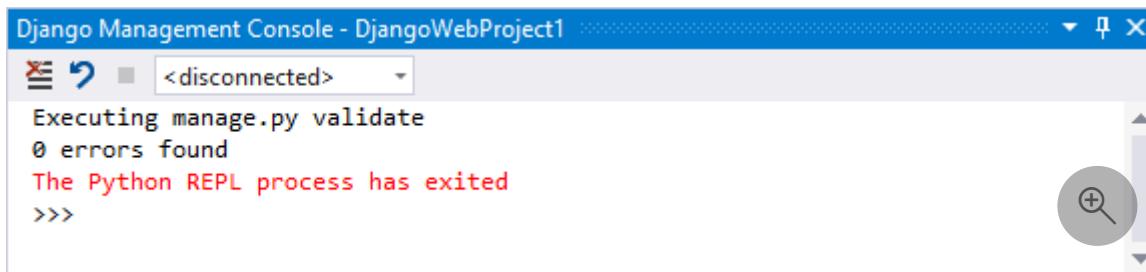
- **Collect Static:** executa o comando `manage.py collectstatic --noinput` para copiar todos os arquivos estáticos para o caminho especificado pela variável `STATIC_ROOT` no arquivo `your_settings.py`.



```
Django Management Console - DjangoWebProject1
disconnected
Executing manage.py collectstatic --noinput

0 static files copied to 'c:\projects\temp2\DjangoWebProject1\static', 91
unmodified.
The Python REPL process has exited
>>> |
```

- **Validate:** executa o comando `manage.py validate`, que relata todos os erros de validação nos modelos instalados especificados pela variável `INSTALLED_APPS` no arquivo `settings.py`:



```
Django Management Console - DjangoWebProject1
disconnected
Executing manage.py validate
0 errors found
The Python REPL process has exited
>>>
```

ⓘ Importante

Projetos Django costumam ser gerenciados por meio de um arquivo `manage.py` e o Visual Studio segue essa abordagem. Caso pare de usar o arquivo `manage.py` como ponto de entrada, você basicamente divide o arquivo de projeto. Nesse caso você

precisa [recriar o projeto usando arquivos existentes](#) sem marcá-lo como um projeto do Django.

Conteúdo relacionado

- Tutorial: Introdução à estrutura da Web do Django no Visual Studio
- Tutorial: usar o modelo Projeto Web do Django completo

Comentários

Esta página foi útil?

 Yes

 No

Criar e gerenciar ambientes Python no Visual Studio

Artigo • 18/04/2024

Um **ambiente de Python** é um contexto em que você executa código Python e inclui ambientes globais, virtuais e conda. Um ambiente é composto por um interpretador, uma biblioteca (normalmente a Biblioteca Padrão do Python) e um conjunto de pacotes instalados. Juntos, esses componentes determinam constructos de linguagem e sintaxe válidos, qual funcionalidade do sistema operacional pode ser acessada e quais pacotes podem ser usados.

No Visual Studio no Windows, use a janela **Ambientes do Python**, conforme descrito neste artigo, para gerenciar ambientes e selecionar um como o padrão para novos projetos. Outros aspectos dos ambientes são encontrados nos seguintes artigos:

- Para qualquer projeto, você pode [selecionar um ambiente específico](#) em vez de usar o padrão.
- Para obter detalhes de como criar e usar ambientes virtuais para projetos Python, confira [Usar ambientes virtuais](#).
- Se desejar instalar pacotes em um ambiente, confira a [referência da guia Pacotes](#).
- Para instalar outro interpretador do Python, confira [Instalar interpretadores do Python](#). Em geral, se você baixar e executar um instalador de uma distribuição principal do Python, o Visual Studio detectará que a nova instalação e o ambiente aparecem na janela **Ambientes do Python** e podem ser selecionados para projetos.

ⓘ Observação

Você pode gerenciar ambientes de código Python abertos como uma pasta ao selecionar **Arquivo > Abrir > Pasta**. A barra de ferramentas do Python permite alternar entre todos os ambientes detectados e também adicionar um novo ambiente. As informações de ambiente são armazenadas no arquivo `PythonSettings.json` na pasta `.vs` do espaço de trabalho.

Pré-requisitos

- Ter uma carga de trabalho do Python instalada.

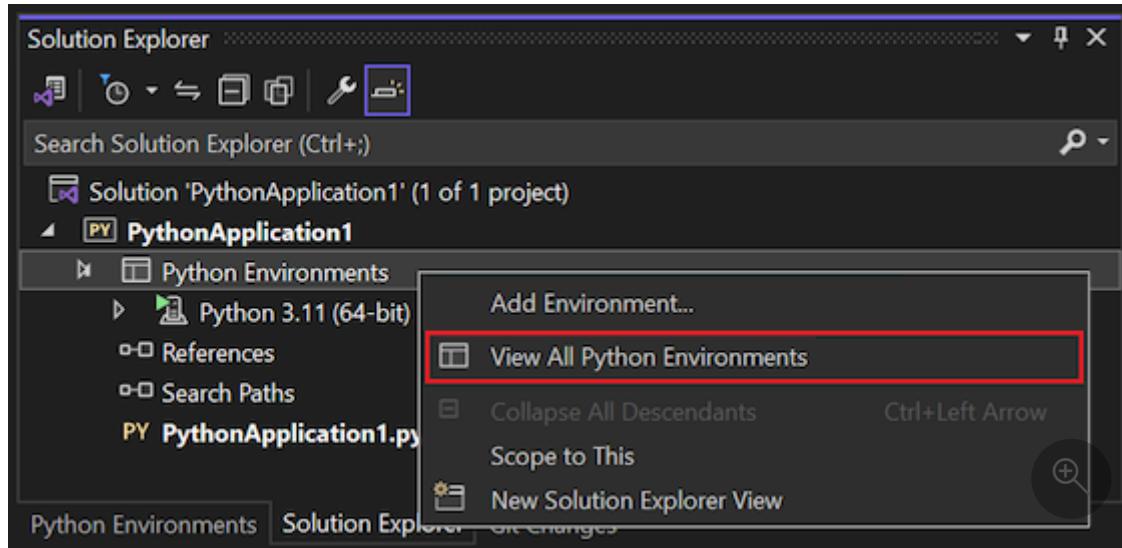
Caso esteja conhecendo agora o Python no Visual Studio, confira os seguintes artigos para obter informações gerais:

- [Trabalhar com o Python no Visual Studio](#)
- [Instalar o suporte de Python no Visual Studio](#)

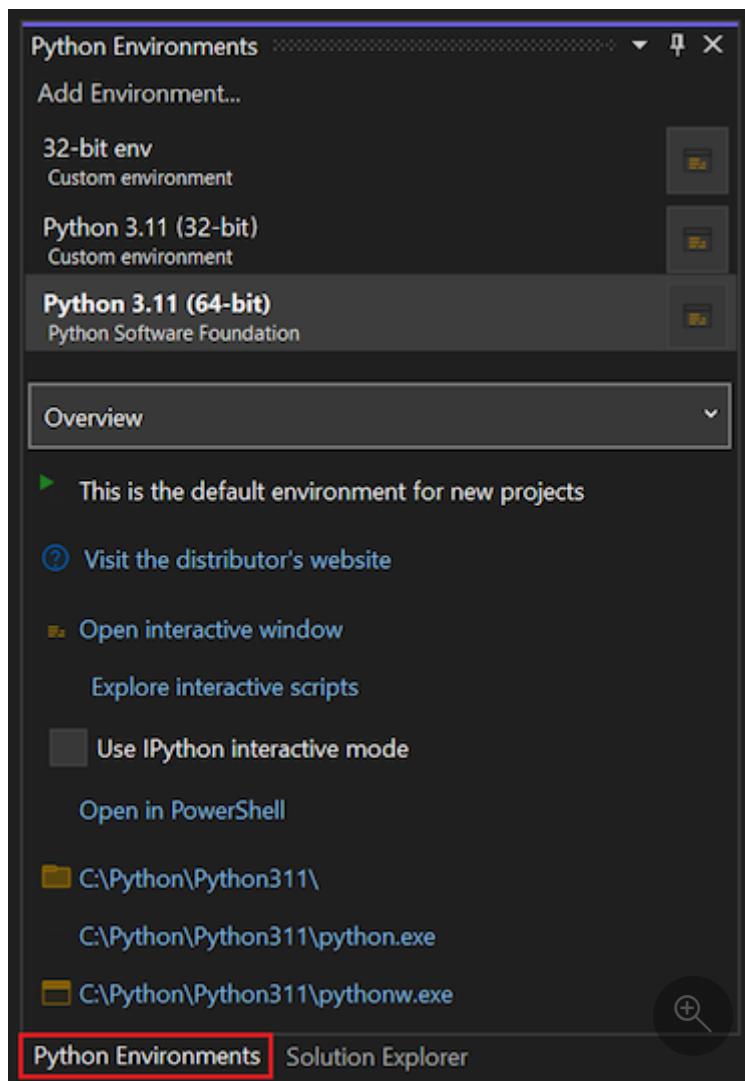
A janela Ambientes do Python

Os ambientes conhecidos pelo Visual Studio são exibidos na janela **Ambientes Python**. Para abrir a janela, use um dos seguintes métodos:

- Escolha **Exibir>Outras Janelas>Ambientes do Python**.
- Clique com o botão direito do mouse no nó **Ambientes do Python** de um projeto no **Gerenciador de Soluções** e escolha **Exibir Todos os Ambientes do Python**.



A janela **Ambientes do Python** é exibida com o **Gerenciador de Soluções** no Visual Studio:



O Visual Studio procura por ambientes globais instalados usando o Registro (seguindo o [PEP 514](#)), junto com os ambientes virtuais e os ambientes do Conda (confira [Tipos de ambientes](#)). Se um ambiente esperado na lista não for exibido, confira [Manually identify an existing environment](#) (Identificar manualmente um ambiente existente).

Quando você escolhe um ambiente da lista, o Visual Studio exibe diversas propriedades e comandos para esse ambiente na guia **Visão Geral** da janela **Ambientes do Python**, como o local do interpretador. Os comandos na parte inferior da guia **Visão Geral** abrem um prompt de comando com o interpretador em execução. Para obter mais informações, confira [Referência de guias da janela Ambientes do Python – Visão Geral](#).

Use a lista suspensa embaixo da lista de ambientes para alternar para diferentes guias como **Pacotes** e **IntelliSense**. Essas guias também são descritas na [Referência de guias da janela Ambientes do Python](#).

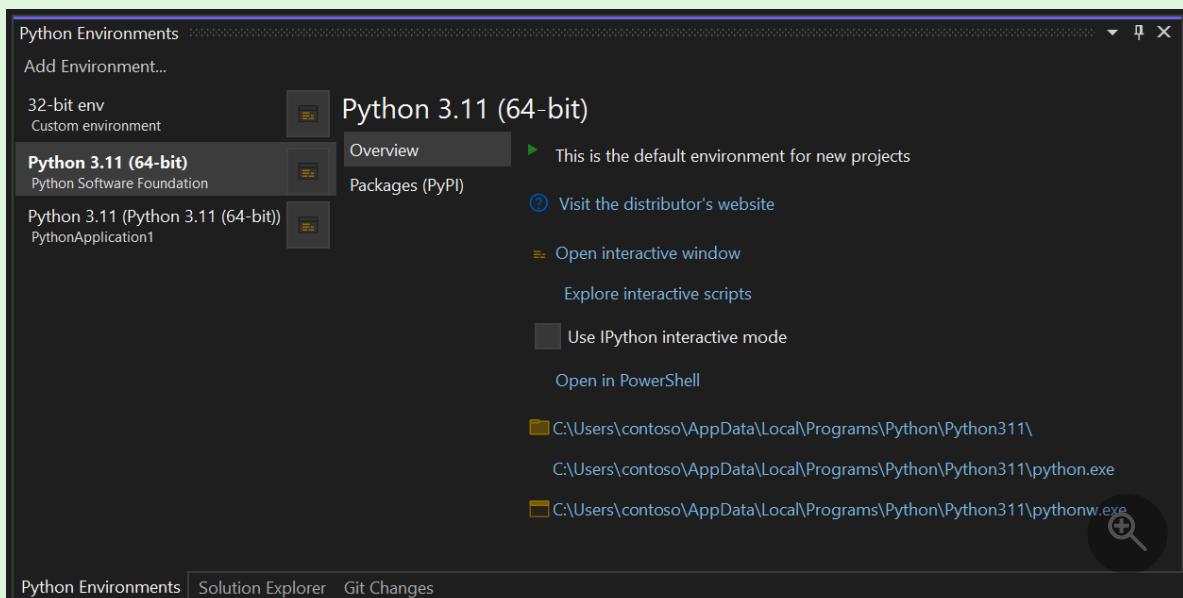
A seleção de um ambiente não altera sua relação com nenhum projeto. O ambiente padrão, mostrado em negrito na lista, é aquele que o Visual Studio usa para os novos projetos. Para usar um ambiente diferente com os novos projetos, use o comando **Tornar este ambiente o padrão para novos projetos**. No contexto de um projeto,

sempre é possível selecionar um ambiente específico. Para obter mais informações, confira [Selecionar um ambiente para um projeto](#).

À direita de cada ambiente listado, há um controle que abre uma janela **Interativa** nesse ambiente. (No Visual Studio 2017 15.5 e versões anteriores, aparece outro controle que atualiza o banco de dados do IntelliSense para esse ambiente. Para obter detalhes sobre o banco de dados, consulte [Referência de guias da janela Ambientes do Python](#).)

💡 Dica

Ao expandir a janela **Ambientes do Python** até o tamanho suficiente, você obtém uma exibição mais completa de seus ambientes, a qual pode ser mais conveniente para trabalhar.



⚠️ Observação

Embora o Visual Studio respeite a opção de pacotes de site do sistema, ele não fornece uma maneira de alterá-lo no próprio Visual Studio.

E se nenhum ambiente aparecer?

Se não aparecer nenhum ambiente na janela **Ambientes do Python**, o Visual Studio não terá conseguido detectar nenhuma instalação do Python nos locais padrão. Talvez você tenha instalado o Visual Studio 2017 ou posterior, mas limpou todas as opções de interpretador nas opções do instalador para a carga de trabalho do Python. Da mesma forma, é possível que você tenha instalado o Visual Studio 2015 ou versões anteriores,

mas não tenha instalado um interpretador manualmente. Para obter mais informações, confira [Instalar interpretadores do Python](#).

Se você souber que tem um interpretador do Python no computador, mas o Visual Studio (qualquer versão) não conseguir detectá-lo, use o comando + Personalizado para especificar o local do interpretador manualmente. Para obter mais informações, confira como [identificar manualmente um ambiente existente](#).

Tipos de ambientes

O Visual Studio pode trabalhar com ambientes globais, virtuais e do Conda.

Ambientes globais

Cada instalação do Python mantém o próprio *ambiente global*. Por exemplo, Python 2.7, Python 3.6, Python 3.7, Anaconda 4.4.0 e assim por diante. Para obter mais informações, confira [Instalar interpretadores do Python](#).

Cada ambiente é composto pelo interpretador do Python específico, sua biblioteca padrão e por um conjunto de pacotes pré-instalados. Também contém todos os outros pacotes instalados enquanto o ambiente está ativado. A instalação de um pacote em um ambiente global o torna disponível para todos os projetos que usam esse ambiente. Se o ambiente estiver em uma área protegida do sistema de arquivos (em *c:\Arquivos de Programas*, por exemplo), a instalação de pacotes exigirá privilégios de administrador.

Os ambientes globais estão disponíveis para todos os projetos no computador. No Visual Studio, selecione um ambiente global como o padrão, que é usado para todos os projetos, a menos que você escolha especificamente um diferente para um projeto. Para obter mais informações, confira [Selecionar um ambiente para um projeto](#).

Ambientes virtuais

Trabalhar em um ambiente global é uma maneira fácil de começar. Com o passar do tempo, os ambientes podem ficar confusos, com muitos pacotes diferentes instalados para projetos diferentes. Essa desordem pode dificultar o teste completo do aplicativo em relação a um conjunto específico de pacotes com versões conhecidas. No entanto, esse tipo de ambiente é o que você configuraria em um servidor de build ou servidor Web. Também poderão ocorrer conflitos quando dois projetos exigirem pacotes incompatíveis ou versões diferentes do mesmo pacote.

Por esses motivos, os desenvolvedores geralmente criam um *ambiente virtual* para um projeto. Um ambiente virtual é uma subpasta em um projeto que contém uma cópia de

um interpretador específico. Se você ativa o ambiente virtual, todos os pacotes passam a ser instalados somente na subpasta desse ambiente. Ao executar um programa em Python dentro do ambiente virtual, você sabe que o programa será executado apenas nesses pacotes específicos.

O Visual Studio dá suporte direto para a criação de um ambiente virtual para um projeto. Caso você abra um projeto que contém um arquivo `requirements.txt`, o Visual Studio envia uma solicitação automática para criar um ambiente virtual e instalar essas dependências. O mesmo comportamento é visto ao criar um projeto com base em um modelo que inclui o arquivo `requirements.txt`.

A qualquer momento dentro de um projeto aberto, você pode criar um ambiente virtual. No **Gerenciador de Soluções**, expanda o nó do projeto, clique com o botão direito do mouse no nó **Ambientes do Python** e escolha **Adicionar ambiente**. Em **Adicionar Ambiente**, escolha **Ambiente virtual**. Para obter mais informações, confira [Criar um ambiente virtual](#).

O Visual Studio também fornece um comando para gerar um arquivo `requirements.txt` em um ambiente virtual, tornando mais fácil recriar o ambiente em outros computadores. Para obter mais informações, confira [Usar ambientes virtuais](#).

Ambientes do Conda

Você pode criar um ambiente do Conda usando a ferramenta `conda`, ou com o gerenciamento do Conda integrado no Visual Studio 2017 versão 15.7 ou posterior. Um ambiente do Conda exige Anaconda ou Miniconda. Essas plataformas são disponibilizadas pelo instalador do Visual Studio. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

1. Na janela **Ambientes do Python** (ou na barra de ferramentas Python), selecione **Adicionar Ambiente** para abrir a caixa de diálogo **Adicionar ambiente**.
2. Na caixa de diálogo **Adicionar ambiente**, selecione a guia **Ambiente do Conda**:

Add environment

Virtual environment
Conda environment
Existing environment
Python installation

Project
PythonApplication1

Name
env

Add packages from

Environment file
environment.yml

One or more Anaconda package names
Example: "numpy pandas".

Set as current environment
 Set as default environment for new projects
 View in Python environments window

Packages preview

| Package | Version |
|-----------------|-------------|
| bzip2 | 1.0.8 |
| ca-certificates | 2021.10.26 |
| certifi | 2020.6.20 |
| libffi | 3.4.2 |
| openssl | 1.1.1l |
| pip | 21.2.4 |
| python | 3.10.0 |
| setuptools | 58.0.4 |
| sqlite | 3.36.0 |
| tk | 8.6.11 |
| tzdata | 2021e |
| vc | 14.2 |
| vs2015_runtime | 14.27.29016 |
| wheel | 0.37.0 |
| wincerstore | 0.2 |
| xz | 5.2.5 |
| zlib | 1.2.11 |

How do I manage Python environments?

Create Cancel

3. Configure os seguintes campos:

Expandir a tabela

| Campo | Descrição |
|-----------------------------|---|
| Projeto | Identifica o projeto em que deseja criar um ambiente. |
| Nome | Informa o nome para o ambiente do Conda. |
| Adicionar pacotes de | <p>Define como adicionar pacotes ao ambiente do Conda.</p> <ul style="list-style-type: none">- Arquivo de ambiente: escolha esta opção caso tenha um arquivo <code>environment.yml</code> que descreva suas dependências. Insira o nome do arquivo ou acesse (...) o local dele e selecione-o.- Um ou mais nomes de pacote do Anaconda: escolha esta opção caso queira listar um ou mais pacotes do Python ou versões do Python. <p>A lista de pacotes instrui o conda a criar um ambiente de Python. Para instalar a versão mais recente do Python, use o comando <code>python</code>. Para instalar uma versão específica, utilize o comando <code>python=<major><minor></code>, como em <code>python=3.7</code>. Você também pode usar o botão de pacote para selecionar as versões do Python e pacotes comuns de uma série de menus.</p> |
| Definir como ambiente atual | Depois que o ambiente for criado, ativa o novo ambiente no projeto selecionado. |
| Definir como ambiente | Define e ativa o ambiente do conda automaticamente em todos os novos projetos criados no Visual Studio. Essa opção é o mesmo que |

| Campo | Descrição |
|---|--|
| padrão para novos projetos | usar Tornar este o ambiente padrão para novos projetos na janela Ambientes do Python. |
| Exibir na janela de ambientes do Python | Especifica se deve mostrar a janela Ambientes do Python depois de criar o ambiente. |

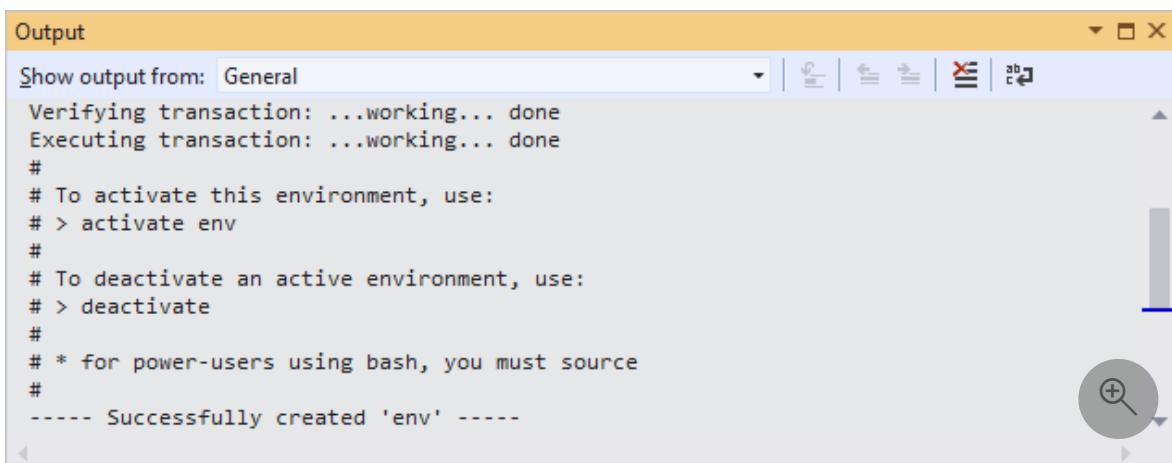
ⓘ Importante

Ao criar um ambiente do Conda, especifique pelo menos uma versão do Python ou um pacote do Python a fim de garantir que o ambiente contenha um tempo de execução do Python. É possível usar um arquivo `environments.yml` ou a lista de pacotes. Se você não informar essa especificação, o Visual Studio vai ignorar o ambiente. O ambiente não aparece em nenhum lugar na janela **Ambientes do Python**, nem está definido como ambiente atual de um projeto, nem está disponível como um ambiente global.

Caso você crie um ambiente do Conda sem uma versão do Python, utilize o comando `conda info` para conferir os locais das pastas do ambiente do Conda. Depois, você poderá remover manualmente a subpasta do ambiente desse local.

4. Selecione Criar.

É possível monitorar a criação do ambiente do Conda na janela **Saída**. Assim que criação for concluída, a saída exibirá algumas instruções da interface de linha de comando (CLI), como `activate env`:



The screenshot shows the Visual Studio Output window with the title bar "Output". The status bar at the bottom right indicates "1 file(s) found". The main area displays the following text:

```
Show output from: General
Verifying transaction: ...working... done
Executing transaction: ...working... done
#
# To activate this environment, use:
# > activate env
#
# To deactivate an active environment, use:
# > deactivate
#
# * For power-users using bash, you must source
#
----- Successfully created 'env' -----
```

5. No Visual Studio, é possível ativar um ambiente do Conda para um projeto da mesma forma que você ativaría qualquer outro ambiente. Para obter mais

informações, confira [Selecionar um ambiente para um projeto](#).

6. Para instalar mais pacotes no ambiente, utilize a guia **Pacotes** na janela **Ambientes do Python**.

ⓘ Observação

Para conseguir os melhores resultados com um ambiente do Conda, utilize o Conda 4.4.8 ou versões posteriores. Lembre-se de que as versões do Conda são diferentes das versões do Anaconda. Você pode instalar versões adequadas do Miniconda (Visual Studio 2019 e Visual Studio 2022) e Anaconda (Visual Studio 2017) pelo instalador do Visual Studio.

Para ver a versão do Conda, na qual os ambientes do Conda são armazenados, bem como outras informações, execute o comando `conda info` em um prompt de comando do Anaconda (ou seja, um prompt de comando no qual o Anaconda está no caminho):

Console

```
conda info
```

As pastas de ambiente do conda são exibidas da seguinte maneira:

Saída

```
envs directories : C:\Users\user\.conda\envs
                  c:\anaconda3\envs
                  C:\Users\user\AppData\Local\conda\conda\envs
```

Como os ambientes do Conda não são armazenados com um projeto, eles se comportam da mesma forma que os ambientes globais. Por exemplo, a instalação de um novo pacote em um ambiente do Conda torna esse pacote disponível para todos os projetos que usam esse ambiente.

Para o Visual Studio 2017 versão 15.6 e anterior, você pode usar ambientes do Conda apontando-os manualmente, conforme descrito em [Identificar manualmente um ambiente existente](#).

O Visual Studio 2017 versão 15.7 e posterior detecta ambientes do conda automaticamente e exibe-os na janela **Ambientes do Python**, conforme descrito na próxima seção.

Identificar manualmente um ambiente existente

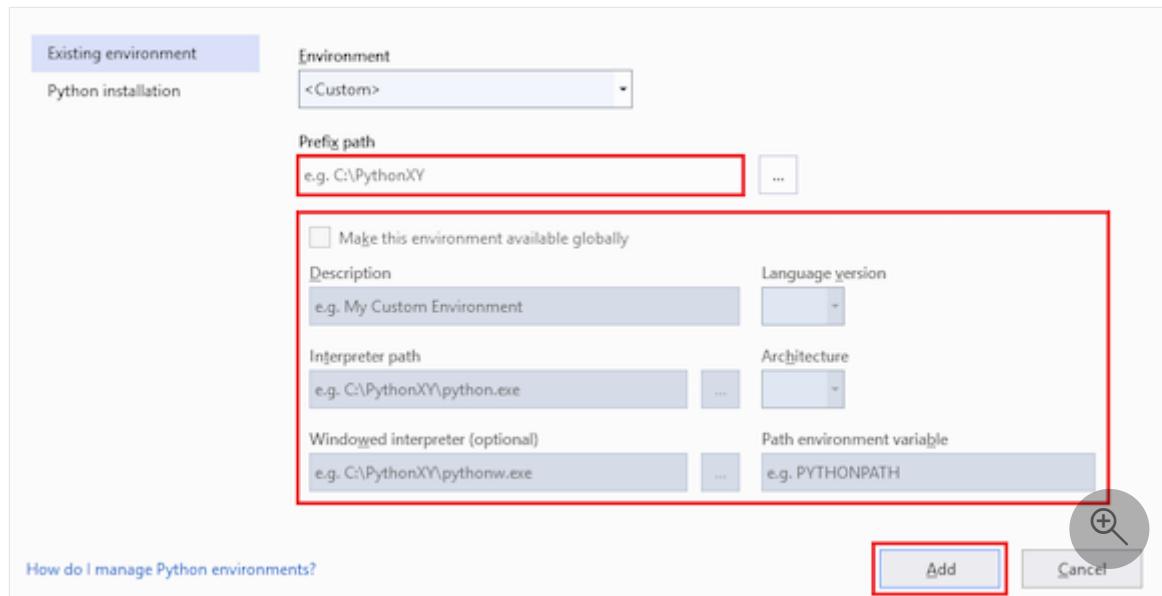
Use as etapas a seguir para identificar um ambiente instalado em um local não padrão.

1. Na janela **Ambientes do Python** (ou na barra de ferramentas Python), selecione **Adicionar Ambiente** para abrir a caixa de diálogo **Adicionar ambiente**.
2. Na guia **Ambiente existente** da caixa de diálogo **Adicionar ambiente**, ajuste o campo **Ambiente** como **<Personalizado>**:



Após selecionar o valor **<Personalizado>** , mais campos serão adicionados à caixa de diálogo.

3. Ajuste o campo **Caminho do prefixo** como o caminho do interpretador. É possível ajustar o campo acessando (...) o local do caminho.



4. Após escolher o caminho, os campos restantes serão preenchidos. Revise os valores e modifique o que for necessário. Quando terminar, selecione **Adicionar**.

Você também pode examinar e modificar detalhes do ambiente a qualquer momento na janela **Ambientes do Python**.

1. Na janela **Ambientes do Python**, selecione o ambiente e escolha a guia **Configurar**.
2. Depois de fazer alterações, selecione o comando **Aplicar**.

Também é possível remover um ambiente usando o comando **Remover**. Para obter mais informações, consulte a guia **Configurar**. Este comando não está disponível para ambientes com detecção automática.

Corrigir ou excluir ambientes inválidos

Se o Visual Studio encontra entradas do Registro para um ambiente, mas o caminho para o interpretador é inválido, a janela **Ambientes do Python** exibe o nome do ambiente em um formato de fonte riscada, conforme mostra a imagem a seguir:



Para corrigir um ambiente que você deseja manter, primeiro tente usar o processo **Reparar** do instalador desse ambiente. Grande parte dos instaladores inclui uma opção para reparar.

Modificar o registro para corrigir um ambiente

Se o ambiente do Python não tiver uma opção para reparar ou se você quiser remover um ambiente inválido, poderá usar as etapas a seguir para modificar o Registro diretamente. O Visual Studio atualiza automaticamente a janela **Ambientes do Python** quando você faz alterações no Registro.

1. Execute o executável `regedit.exe` para abrir o Editor do Registro.

2. Acesse a pasta do ambiente correspondente à sua configuração:

 Expandir a tabela

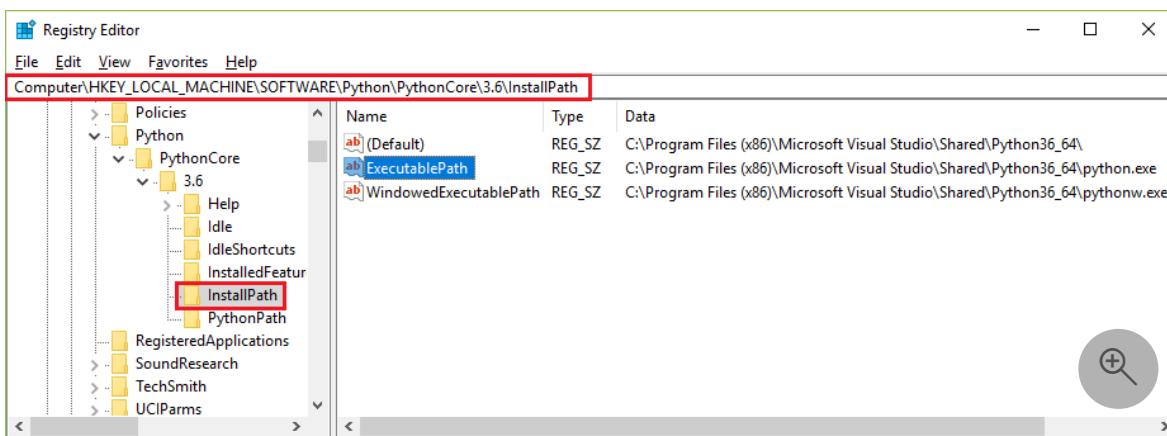
| Versão do Python | Pasta |
|-------------------|---|
| Versão de 64 bits | HKEY_LOCAL_MACHINE\SOFTWARE\Python or HKEY_CURRENT_USER\Software\Python |
| Versão de 32 bits | HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Python |
| IronPython | IronPython |

3. Expanda a estrutura de nós de distribuições e versões do ambiente:

 Expandir a tabela

| Distribuição | Nó |
|--------------|-----------------------------------|
| CPython | PythonCore><Nó da versão> |
| Anaconda | ContinuumAnalytics><Nó da versão> |
| IronPython | <Nó da versão> |

4. Inspecione os valores no nó **InstallPath**:



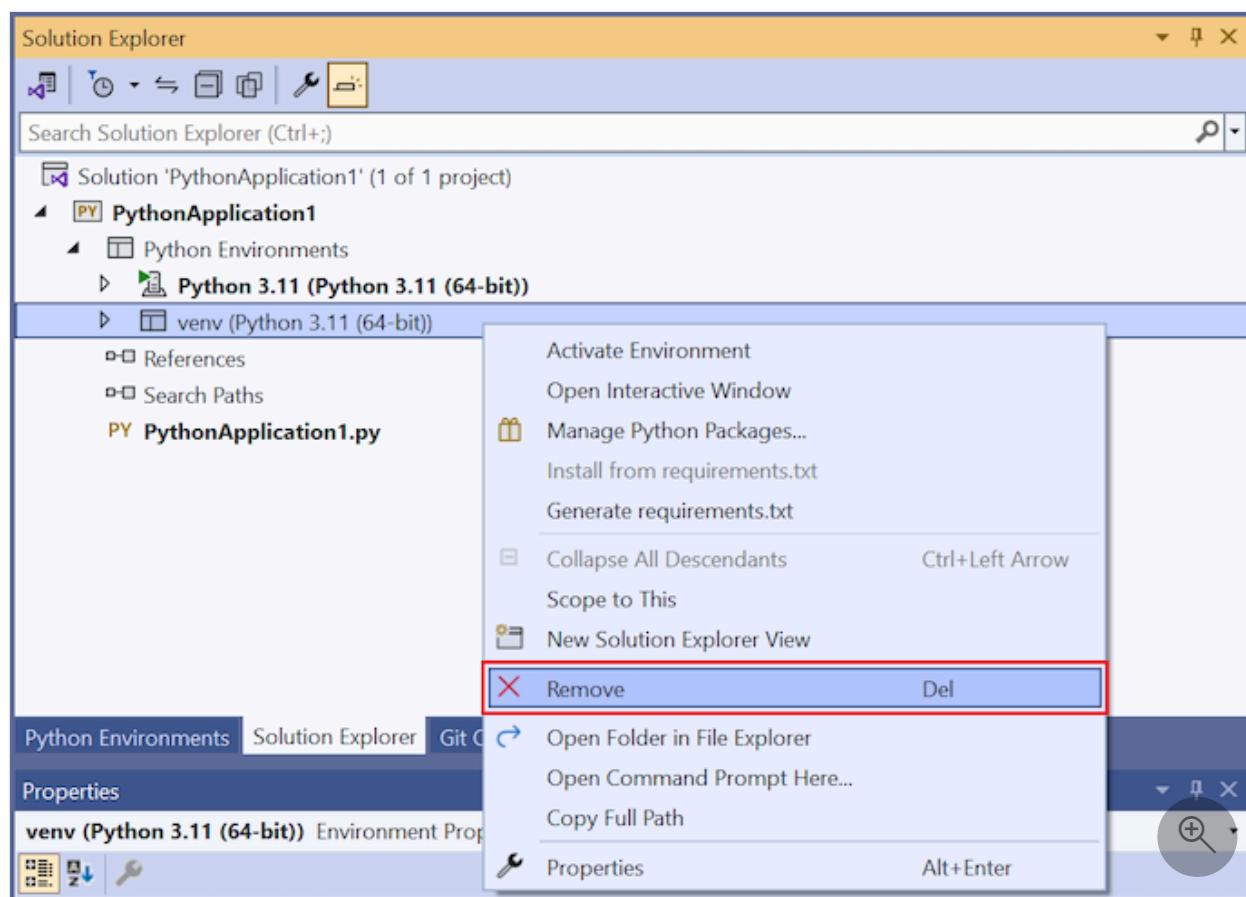
- Se o ambiente ainda existir no computador, altere o valor da entrada **ExecutablePath** para o local correto. Corrija também os valores das entradas (**Padrão**) e **WindowedExecutablePath**, conforme necessário.
- Se o ambiente não existir mais no computador e você desejar removê-lo da janela **Ambientes do Python**, exclua o nó pai do número da versão do nó de **InstallPath**. A imagem anterior apresenta um exemplo desse nó. No exemplo, esse nó é **3.6**.

⊗ Cuidado

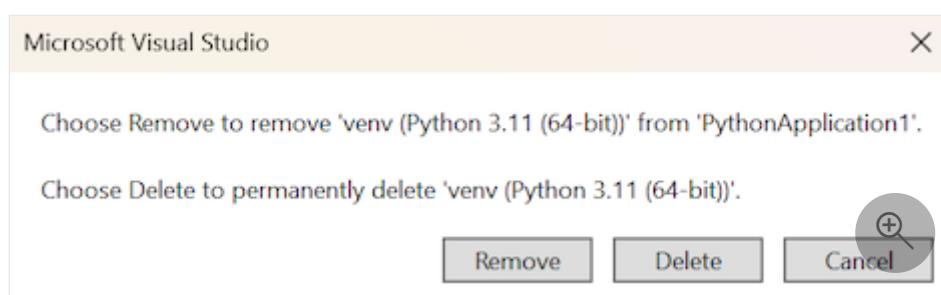
Configurações inválidas na chave **HKEY_CURRENT_USER\SOFTWARE\Python** substituem as configurações na chave **HKEY_LOCAL_MACHINE\SOFTWARE\Python**.

Excluir ou remover um ambiente Python

Se quiser remover um projeto do Python, acesse o ambiente do Python no **Gerenciador de Soluções**. Clique com o botão direito do mouse no ambiente do Python que deseja remover e selecione **Remover**.



Caso queira manter o ambiente do Python e apenas removê-lo de um projeto, escolha **Remover**. Se preferir excluir o ambiente de forma permanente, escolha **Excluir**.



Conteúdo relacionado

- Instalar interpretadores do Python
 - Selecionar um interpretador para um projeto
 - Use requirements.txt para dependências
 - Caminhos de pesquisa
 - Referência da janela de ambientes do Python
-

Comentários

Esta página foi útil?

 Yes

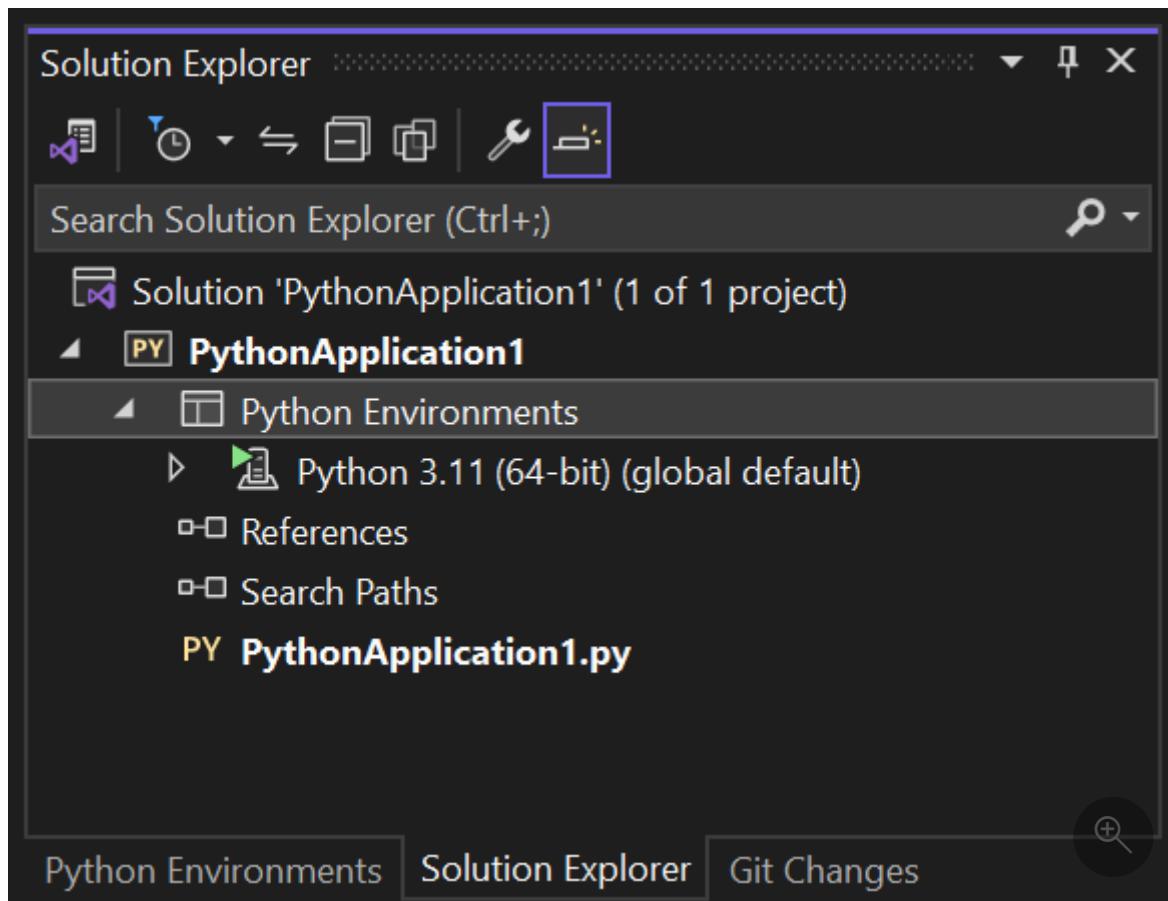
 No

Selecionar um ambiente Python para um projeto no Visual Studio

Artigo • 18/04/2024

Todo o código em um projeto do Python é executado dentro do contexto de um ambiente específico. Esses ambientes podem ser um ambiente global do Python, um ambiente do Anaconda, um ambiente virtual ou um ambiente do Conda. O Visual Studio utiliza o ambiente do Python para depuração, importação, conclusão de membros e verificação de sintaxe. O ambiente é utilizado para todas as tarefas que exijam serviços de linguagem específicos da versão do Python e um conjunto de pacotes instalados.

No Visual Studio, é possível criar vários ambientes para um projeto e alternar entre eles conforme as suas necessidades específicas de desenvolvimento. Novos projetos do Python são inicialmente configurados para utilizar o ambiente global padrão. É possível conferir os ambientes de um projeto no nó **Ambientes do Python** no Gerenciador de Soluções:



Pré-requisitos

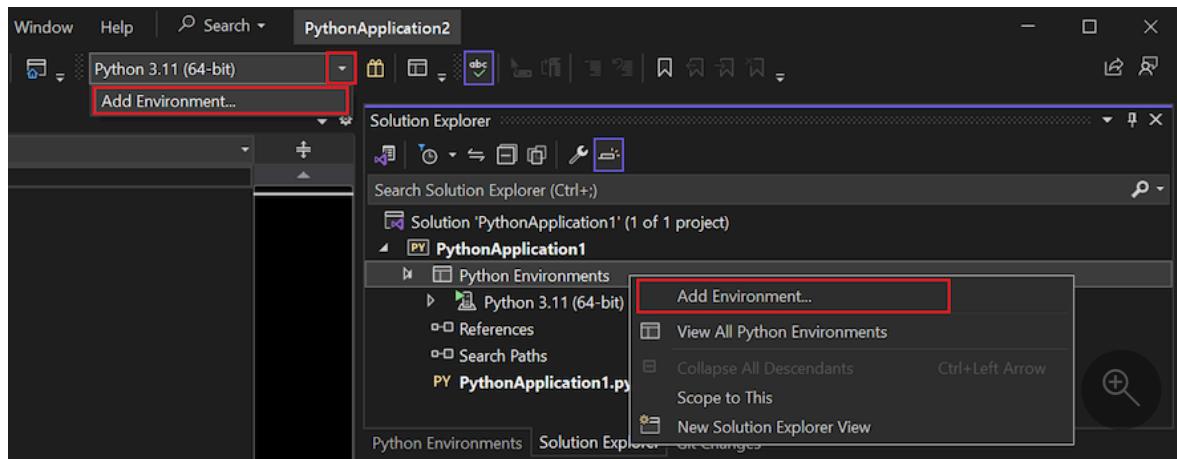
- Visual Studio. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).
- Só é possível alternar entre ambientes existentes. Caso você não tenha outro ambiente além do ambiente global padrão, consulte as seções a seguir para aprender a trabalhar com [ambientes virtuais](#). Para obter mais informações, consulte [Criar e gerenciar ambientes do Python no Visual Studio](#).

Trocar o ambiente do projeto atual

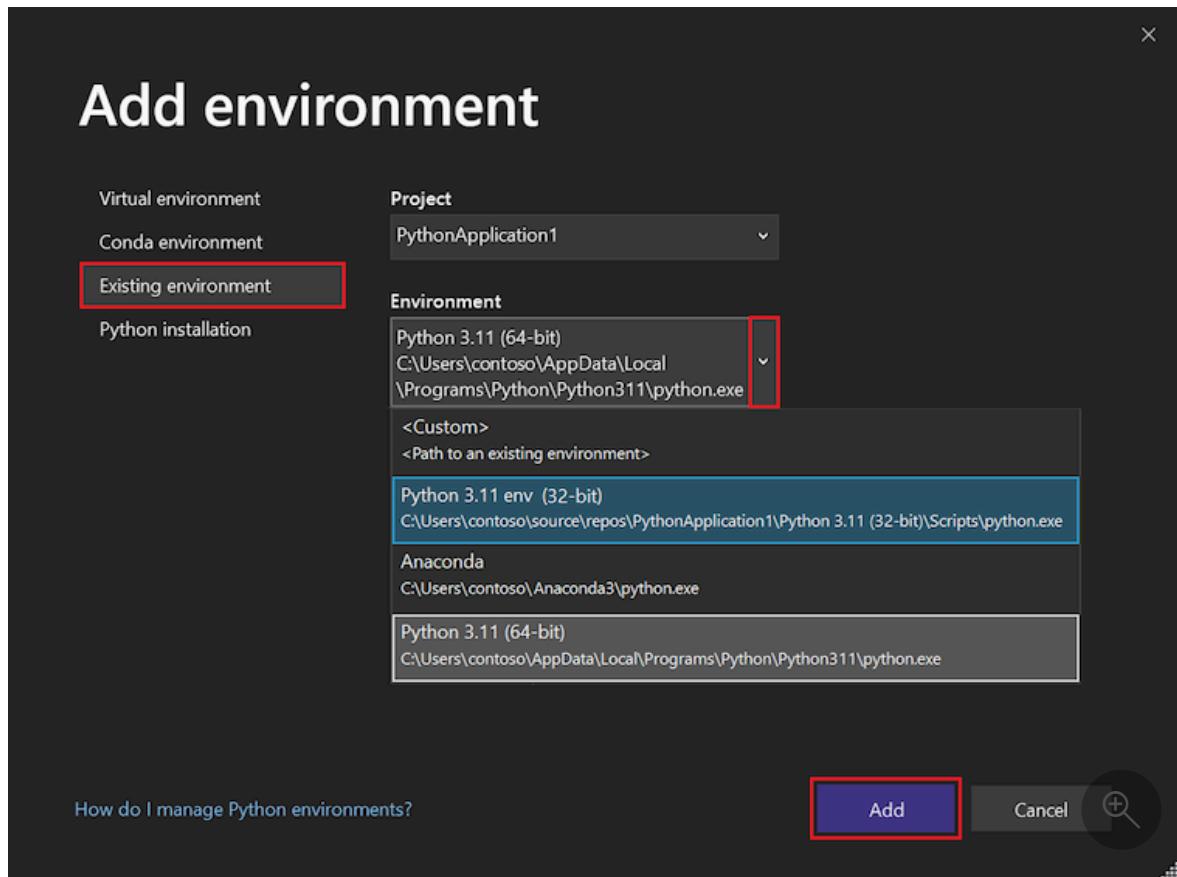
No Visual Studio, é possível trocar o ambiente ativo (atual) de um projeto do Python pelo **Gerenciador de Soluções** ou pela barra de ferramentas utilizando a funcionalidade **Adicionar Ambiente**.

1. Inicie o processo para Adicionar Ambiente:

- No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Ambientes do Python** referente ao seu projeto e escolha **Adicionar Ambiente**.
- Como alternativa, na barra de ferramentas do Python, selecione **Adicionar Ambiente** no menu suspenso **Ambiente**.



2. Na caixa de diálogo **Adicionar Ambiente**, escolha a guia **Ambiente existente**. Expanda a lista suspensa **Ambiente**, selecione um ambiente e escolha **Adicionar**.



⚠️ Observação

Se o ambiente que você quiser utilizar não estiver listado, poderá ser necessário [identificar manualmente um ambiente existente](#).

Usar ambientes virtuais

Um ambiente virtual é uma combinação exclusiva de um interpretador específico do Python e um conjunto específico de bibliotecas que é diferente de outros ambientes globais e do Conda. Um ambiente virtual é específico para um projeto e é mantido em uma subpasta do projeto. A pasta contém as bibliotecas instaladas do ambiente e um arquivo `pyvenv.cfg` que especifica o caminho do *interpretador de base* do ambiente no sistema de arquivos. (Um ambiente virtual não contém uma cópia do interpretador, apenas um link para ele.)

Um dos benefícios do uso de um ambiente virtual é que, à medida que você desenvolve um projeto, o ambiente virtual sempre reflete as dependências exatas do projeto. Esse comportamento é diferente de um ambiente global compartilhado, que contém qualquer quantidade de bibliotecas, independentemente de você utilizá-las ou não no projeto. Em um ambiente virtual, é possível criar facilmente um arquivo `requirements.txt`, que é usado para reinstalar as dependências do pacote em outros computadores de

desenvolvimento ou produção. Para saber mais, confira [Gerenciar pacotes necessários com requirements.txt](#).

Quando você abre um projeto no Visual Studio que contém um arquivo *requirements.txt*, o Visual Studio oferece automaticamente a opção de recriar o ambiente virtual. Em computadores em que o Visual Studio não está instalado, você pode usar o comando `pip install -r requirements.txt` para restaurar os pacotes necessários.

Como um ambiente virtual contém um caminho embutido em código para o interpretador de base do Python e é possível recriar o ambiente usando o arquivo *requirements.txt*, normalmente você omite a subpasta do ambiente no controle do código-fonte. Depois de adicionar um ambiente virtual ao projeto, ele será exibido na janela **Ambientes do Python**. Você pode ativá-lo como qualquer outro ambiente e gerenciar seus pacotes.

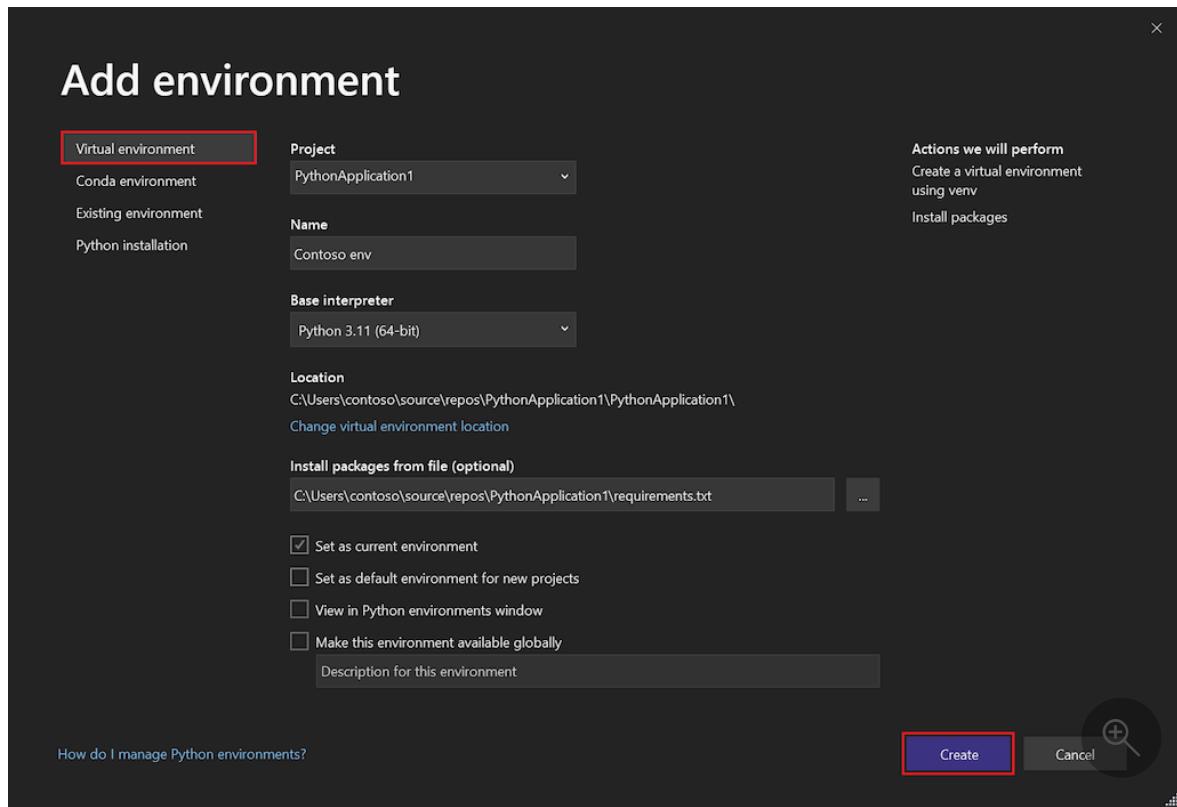
Criar um ambiente virtual

Você pode criar um novo ambiente virtual diretamente no Visual Studio da seguinte maneira:

1. Inicie o processo para **Adicionar Ambiente**:

- No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Ambientes do Python** referente ao seu projeto e escolha **Adicionar Ambiente**.
- Como alternativa, na barra de ferramentas do Python, selecione **Adicionar Ambiente** no menu suspenso **Ambiente**.

2. Na caixa de diálogo **Adicionar Ambiente**, selecione a guia **Ambiente virtual**:



3. Defina os campos obrigatórios:

[] Expandir a tabela

| Campo obrigatório | Descrição |
|-----------------------|---|
| Projeto | Identifique o projeto em que deseja criar um ambiente. |
| Nome | Informe um nome para o novo ambiente virtual. |
| Interpretador de base | Determine o interpretador da linguagem de base do ambiente virtual. |
| Localidade | O local padrão é atribuído ao ambiente virtual pelo sistema. Se quiser trocar o local, escolha o link Alterar local do ambiente virtual , acesse o local e selecione Selecionar pasta . |

4. Defina todos os campos opcionais desejados:

[] Expandir a tabela

| Campo opcional | Descrição |
|-----------------------------------|--|
| Instalar pacotes usando o arquivo | Determine o caminho para um arquivo <i>requirements.txt</i> a fim de adicionar pacotes ao ambiente virtual. Insira o local e o nome do arquivo ou acesse (...) o local dele e selecione-o. |

| Campo opcional | Descrição |
|---|--|
| Definir como ambiente atual | Depois que o ambiente for criado, ative o novo ambiente no projeto selecionado. |
| Definir como ambiente padrão para novos projetos | Defina e ative o ambiente automaticamente em todos os novos projetos criados no Visual Studio. Essa configuração também está disponível pela opção Tornar este o ambiente padrão para novos projetos na janela Ambientes do Python . Ao usar essa opção, coloque o ambiente virtual em um local fora de um projeto específico. |
| Exibir na janela de ambientes do Python | Especifique se deseja mostrar a janela Ambientes do Python depois de criar o ambiente. |
| Tornar esse ambiente disponível globalmente | Especifique se o ambiente virtual também deve atuar como um ambiente global. Ao usar essa opção, coloque o ambiente virtual em um local fora de um projeto específico. |

5. Selecione **Criar** para finalizar o ambiente virtual.

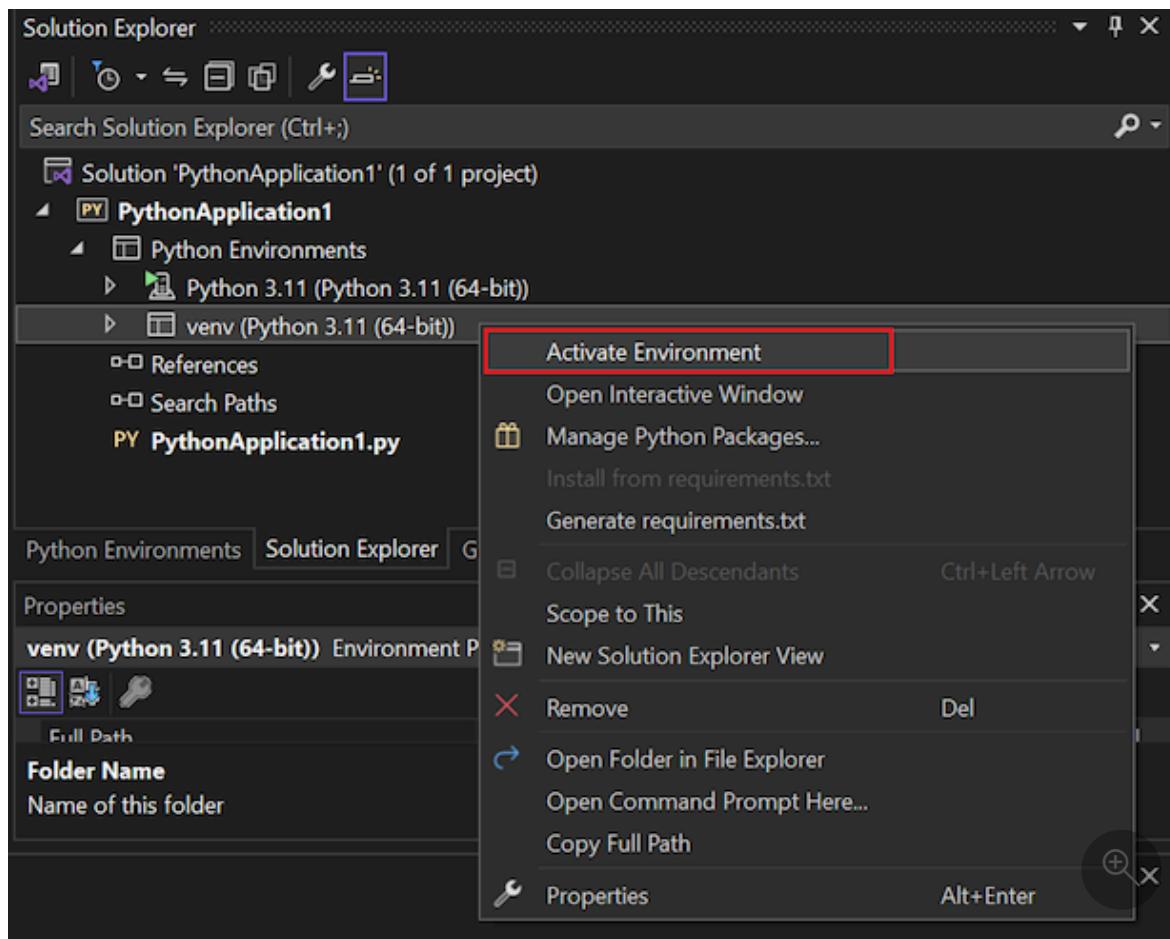
O Visual Studio exibe uma barra de progresso enquanto configura o ambiente, e baixa todos os pacotes necessários.

Depois que o processo for concluído, o Visual Studio ativará o novo ambiente virtual e o adicionará ao nó **Ambientes do Python** no **Gerenciador de Soluções**. O ambiente também pode ser encontrado na janela **Ambientes do Python** para o projeto contido.

Ativar um ambiente

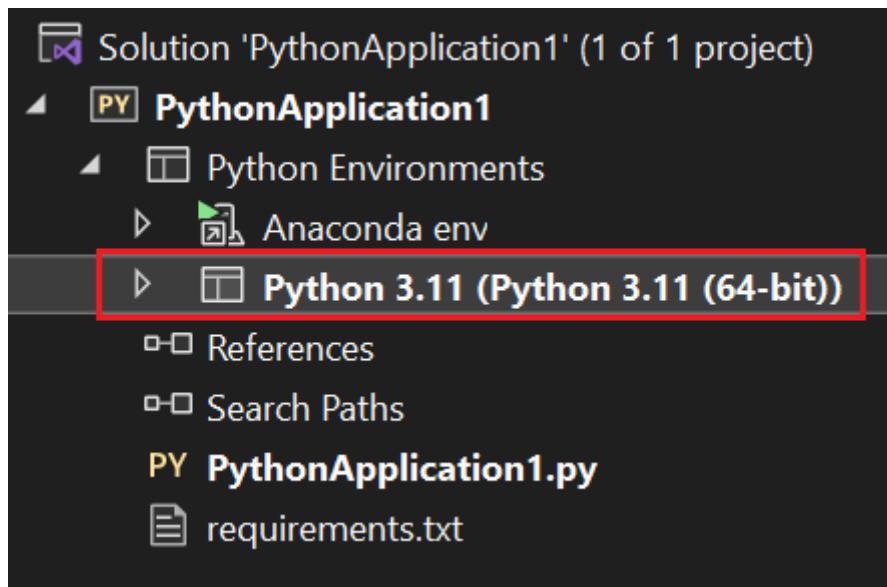
Para ativar um ambiente existente em um projeto, realize as seguintes etapas:

1. No **Gerenciador de Soluções**, expanda o nó **Ambientes do Python** do projeto e encontre o ambiente que deseja utilizar.
2. Clique com o botão direito do mouse no ambiente e selecione **Ativar Ambiente**.



Se o Visual Studio detectar um arquivo *requirements.txt* nesse ambiente, ele perguntará se deve instalar esses pacotes.

Após a ativação do ambiente pelo Visual Studio, o nome do ambiente ativo aparece em negrito no Gerenciador de Soluções:



Remover um ambiente virtual

Para remover um ambiente existente em um projeto, realize as seguintes etapas:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no ambiente virtual e escolha **Remover**.

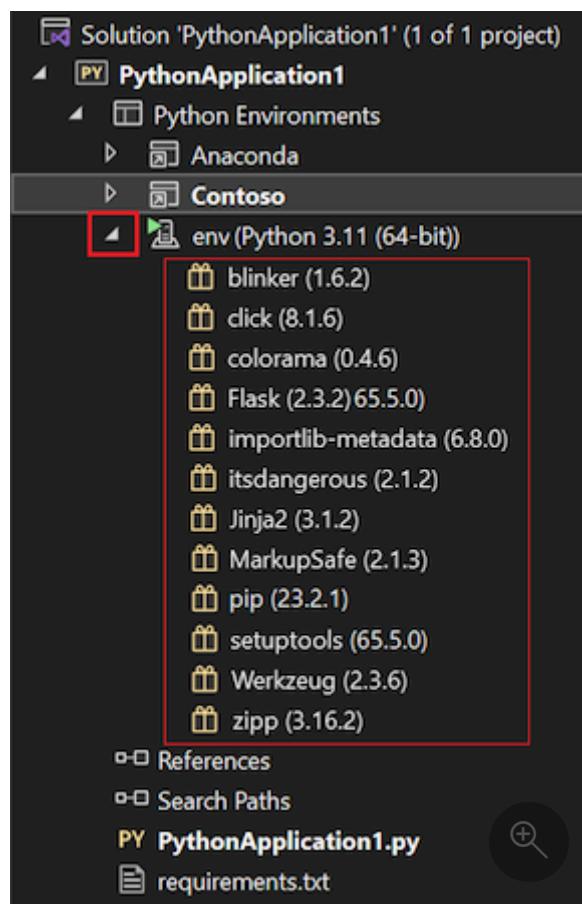
2. O Visual Studio pergunta se você quer remover ou excluir o ambiente virtual.

- Selecione **Remover** para tornar o ambiente indisponível ao projeto, mas mantê-lo no sistema de arquivos.
- Selecione **Excluir** para remover o ambiente do projeto e excluí-lo do sistema de arquivos. O interpretador de base não é afetado.

Visualizar e gerenciar os pacotes instalados

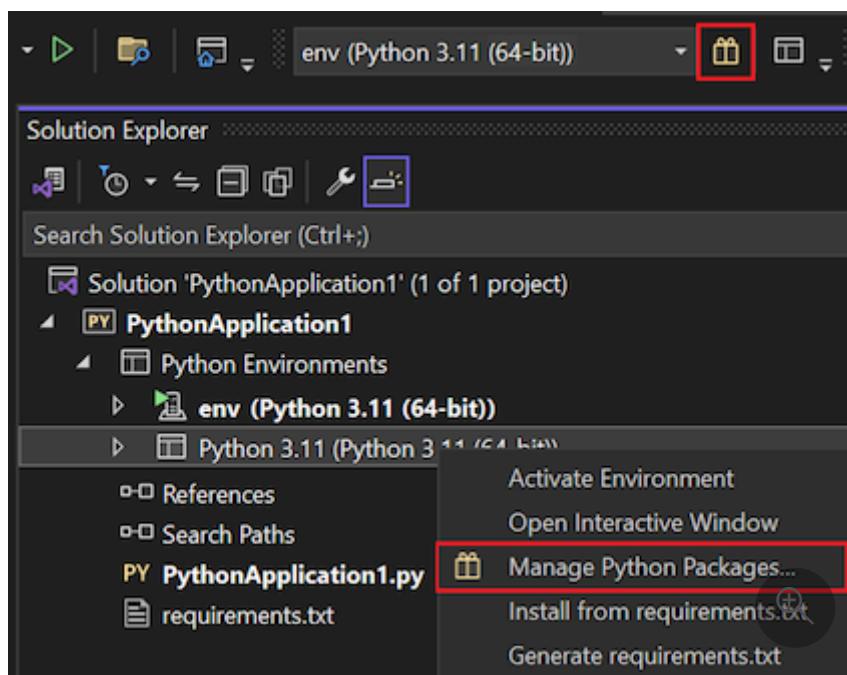
No Gerenciador de Soluções, é possível visualizar e gerenciar todos os pacotes instalados em um ambiente. Você pode importar esses pacotes e utilizá-los no código quando o ambiente estiver ativo.

- Para visualizar rapidamente todos os pacotes instalados em um ambiente, expanda o nó do ambiente no nó **Ambientes do Python** do respectivo projeto no **Gerenciador de Soluções**:

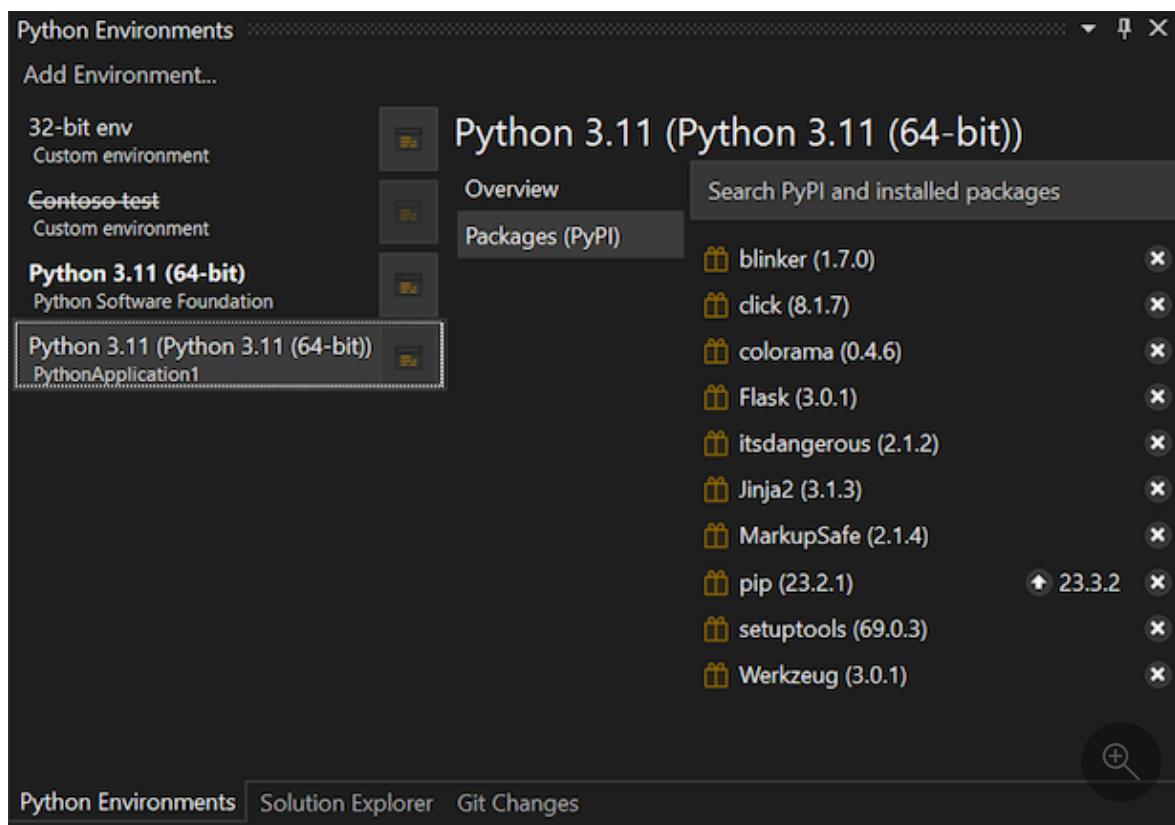


- Se quiser instalar novos pacotes ou gerenciar pacotes existentes, clique com o botão direito do mouse no nó do ambiente e escolha **Gerenciar Pacotes do**

Python. Também é possível utilizar o botão de pacotes na barra de ferramentas do Python:



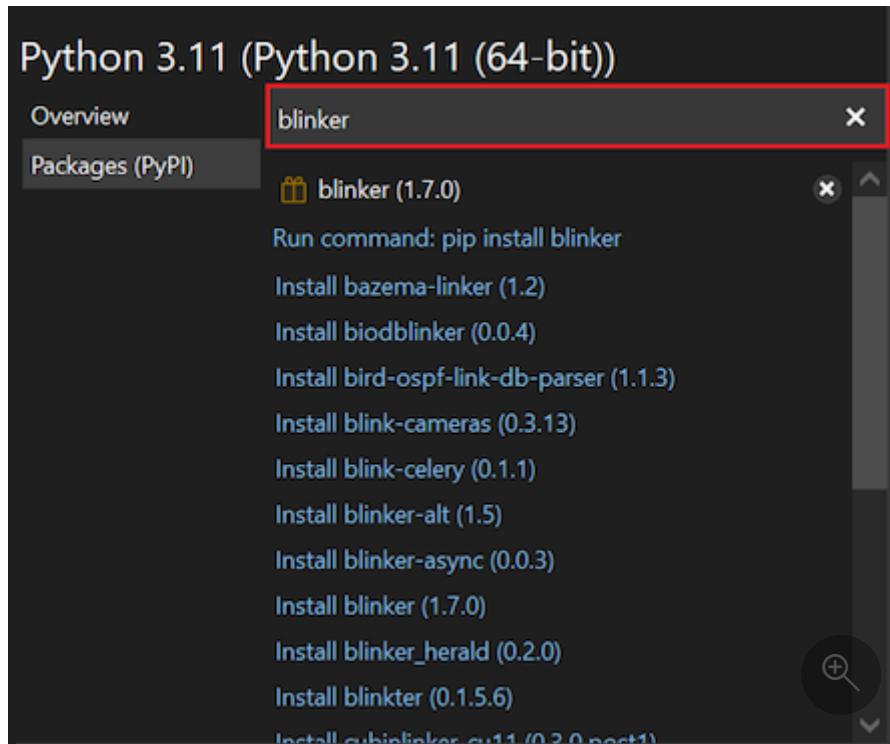
A janela **Ambientes do Python** é exibida, mostrando todos os pacotes instalados no ambiente selecionado na guia **Pacotes (PyPI)**:



No Visual Studio, os pacotes e as dependências da maioria dos ambientes são baixados do [Índice de Pacotes do Python \(PyPI\)](#), onde você também pode pesquisar os pacotes disponíveis. A barra de status e a janela de saída do Visual Studio mostram informações sobre a instalação.

- Se quiser desinstalar (remover) um pacote, encontre-o na lista e escolha o ícone de x à direita dele.
- Para conferir se há versões atualizadas de um pacote ou pesquisar outros pacotes, insira um termo de pesquisa (em geral, um nome de pacote):

O Visual Studio exibirá todos os pacotes correspondentes. Este exemplo mostra uma busca por pacotes com o termo *blinker*.



O Visual Studio exibe uma lista de todos os resultados correspondentes na forma de links de comando ativos.

- O primeiro comando atualiza o pacote para a versão e as dependências mais recentes. O comando é semelhante a `Run command: pip install <package-name>`. Se você pressionar **Enter** depois do termo da pesquisa, o Visual Studio executará automaticamente o primeiro comando.
- Os outros links referem-se a comandos que instalam um pacote, uma versão ou uma dependência específica, como `Install blinker-async (0.0.3)`. Escolha o respectivo link para executar um desses comandos.

Considerações sobre a instalação de pacotes

Ao trabalhar com pacotes no Visual Studio, lembre-se das seguintes considerações:

- As entradas exibidas para os pacotes podem não ser precisas no que se refere à versão mais recente ou à disponibilidade. As informações de instalação e

desinstalação exibidas para determinado pacote podem não ser confiáveis ou podem não estar disponíveis.

- O Visual Studio usa o gerenciador de pacotes do PIP, se disponível, e o baixa e o instala quando necessário. O Visual Studio também pode usar o gerenciador de pacotes easy_install. Os pacotes instalados usando os comandos `pip` ou `easy_install` por meio da linha de comando também são exibidos.
- Uma situação comum em que o PIP não conseguirá instalar um pacote ocorrerá quando o pacote incluir o código-fonte para componentes nativos em arquivos `*.pyd`. Sem a versão necessária do Visual Studio instalada, o PIP não pode compilar esses componentes. A mensagem de erro exibida nessa situação é **erro: Não foi possível localizar vcvarsall.bat**. O comando `easy_install` geralmente consegue baixar os binários pré-compilados, e você pode baixar um compilador adequado para versões mais antigas do Python em <https://python.en.uptodown.com/windows/versions>. Para obter mais informações, consulte [Como lidar com o problema “Não é possível localizar vcvarsallbat”](#) no blog da equipe das ferramentas do Python.
- O gerenciador de pacotes conda geralmente usa <https://repo.continuum.io/pkgs/> como padrão de canal, mas há outros canais disponíveis. Para obter mais informações, consulte [Gerenciar Canais](#) (docs.conda.io).
- O Visual Studio ainda não oferece suporte ao uso do comando `conda` para instalar pacotes em um ambiente do Conda. Em vez disso, use o comando `conda` da linha de comando.

Conteúdo relacionado

- [Gerenciar ambientes do Python no Visual Studio](#)
- [Usar requirements.txt para dependências](#)
- [Caminhos de pesquisa](#)
- [Referência da janela de ambientes do Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Gerenciar os pacotes do Python necessários com requirements.txt

Artigo • 18/04/2024

Caso você compartilhe um projeto do Python com outras pessoas ou use um sistema de compilação com o intuito de produzir o aplicativo em Python, precisará especificar todos os pacotes externos necessários. Ao planejar copiar o projeto para outros locais necessários para restaurar um ambiente, também é preciso definir os pacotes dependentes necessários.

A abordagem recomendada para especificar pacotes externos dependentes do Python é o uso de um [arquivo de requisitos](#) (readthedocs.org). Esse arquivo oferece uma lista de comandos do PIP que instalam todas as versões necessárias dos pacotes dependentes para o projeto. O comando mais comum é `pip freeze > requirements.txt`. Esse comando registra a lista de pacotes atuais do ambiente no arquivo `requirements.txt`.

Um arquivo de requisitos contém versões precisas de todos os pacotes instalados. É possível utilizar arquivos de requisitos para congelar os requisitos de um ambiente. Ao utilizar versões precisas dos pacotes, você conseguirá reproduzir facilmente seu ambiente em outra máquina. O arquivo de requisitos inclui até os pacotes instalados com uma série de versões, como uma dependência de outro pacote ou com um instalador que não seja o PIP.

Pré-requisitos

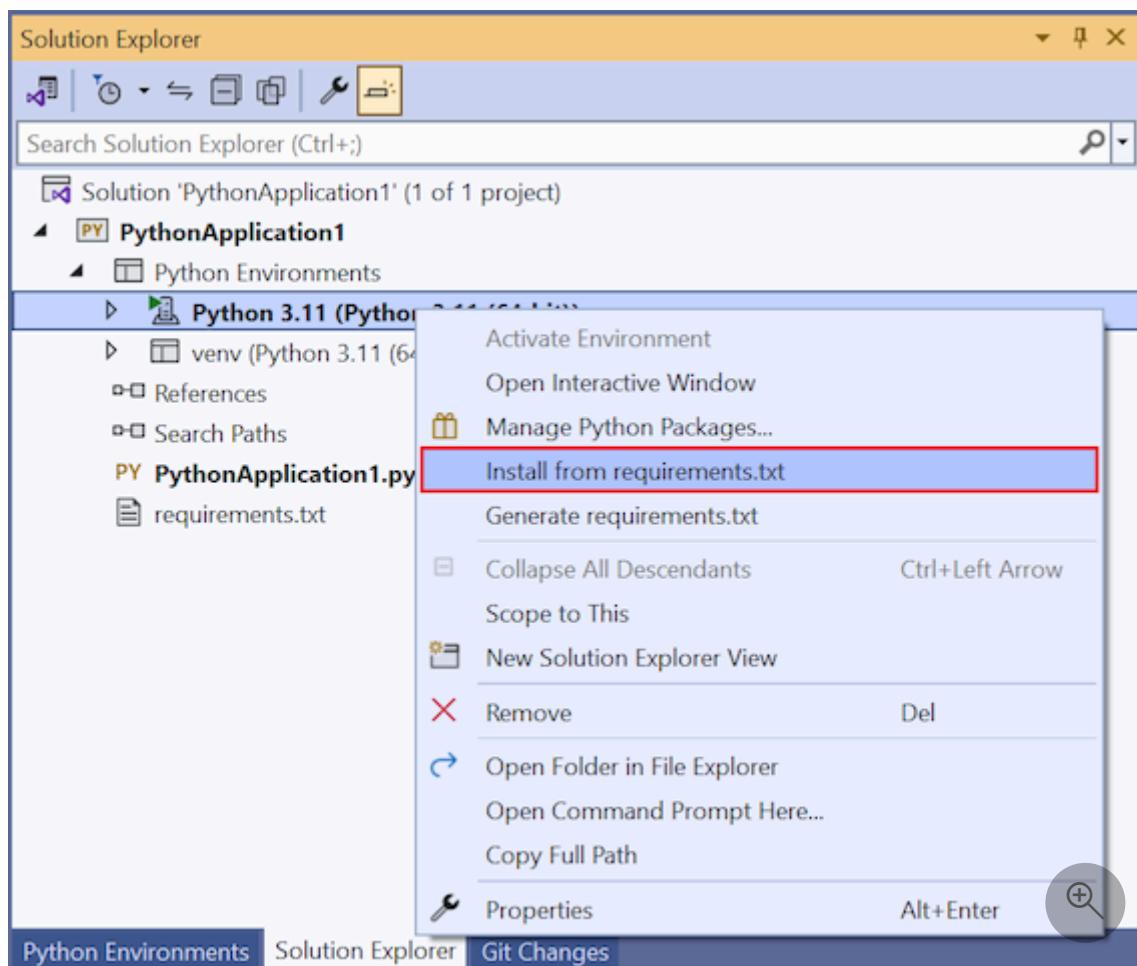
- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Um arquivo de requisitos. É possível utilizar um arquivo de requisitos existente ou [gerar um arquivo](#), conforme as instruções descritas neste artigo.

Tecnicamente, é possível utilizar qualquer nome de arquivo para rastrear os requisitos. Entretanto, o Visual Studio oferece suporte específico ao arquivo de requisitos chamado "requirements.txt". Utilize o argumento `-r <full path to file>` ao instalar um pacote com o intuito de especificar um nome de sua preferência para o arquivo.

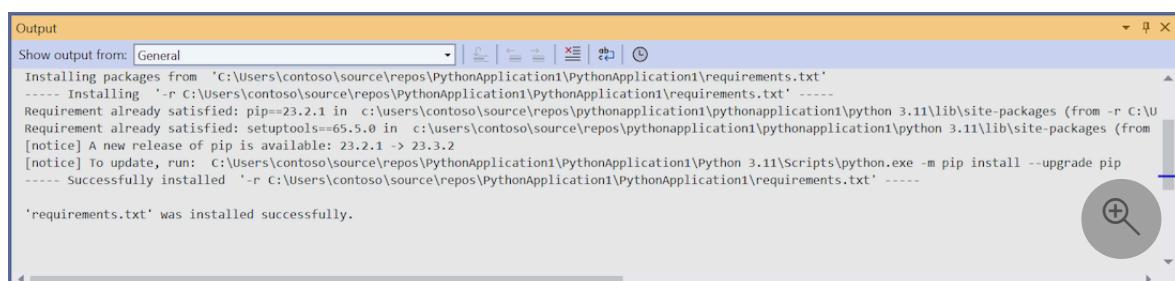
Instalar as dependências listadas em requirements.txt

Caso carregue um projeto com um arquivo `requirements.txt`, você pode instalar todas as dependências de pacote listadas no arquivo.

1. No Gerenciador de Soluções, expanda o projeto, depois expanda o nó Ambientes do Python.
 2. Encontre o nó do ambiente no qual deseja instalar os pacotes. Clique com o botão direito do mouse no nó e escolha **Instalar do requirements.txt**.



3. Você pode monitorar o processo de instalação dos pacotes na janela **Saída**:



A saída lista todos os pacotes necessários que foram instalados e todas as atualizações necessárias para os comandos do PIP afetados, bem como a disponibilidade de versões do PIP mais recentes.

Instalar as dependências em um ambiente virtual

Também é possível instalar as dependências de pacotes do Python em um ambiente virtual existente.

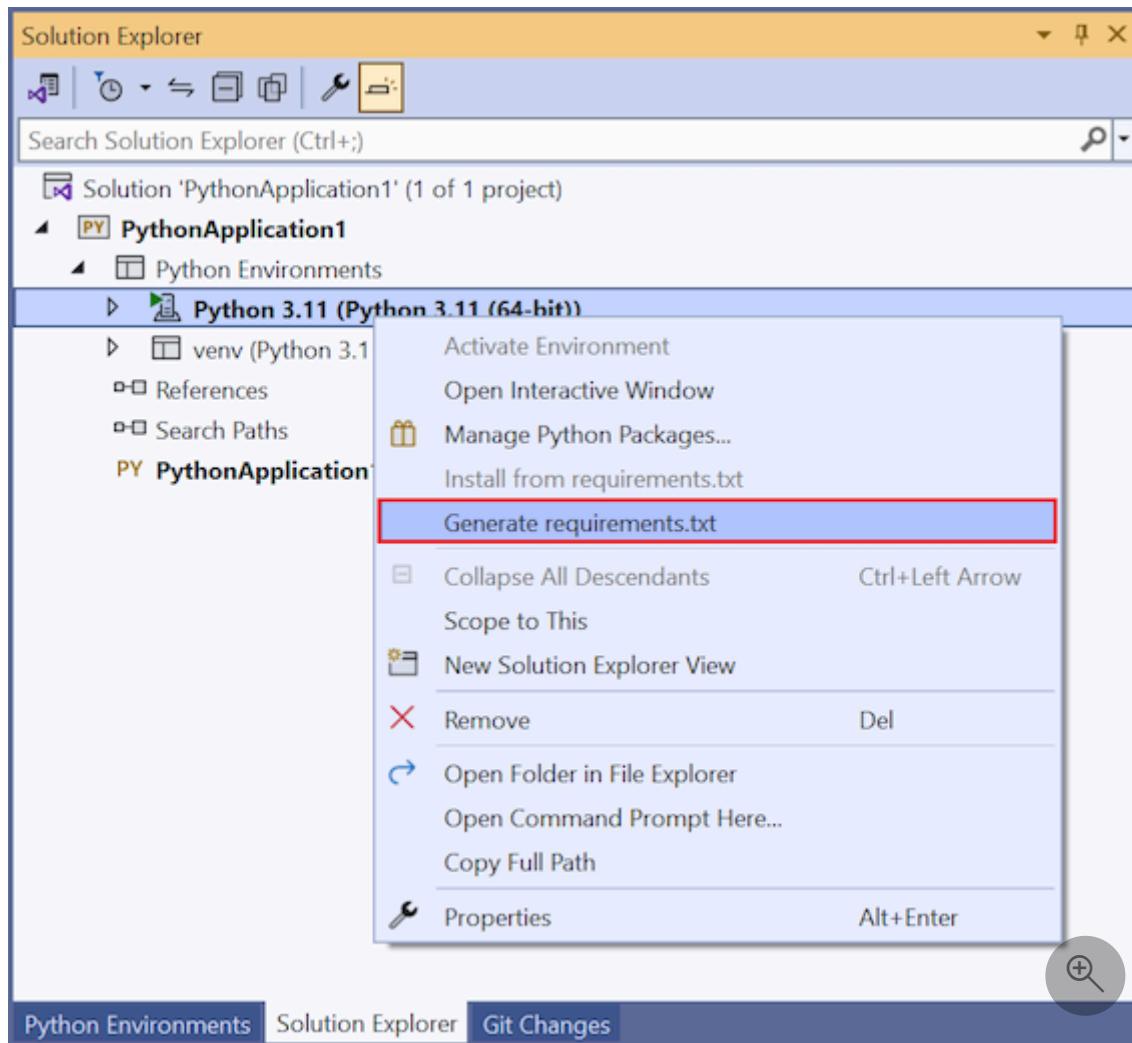
1. No **Gerenciador de Soluções**, expanda o projeto, depois expanda o nó **Ambientes do Python**.
2. Encontre o nó do ambiente virtual no qual deseja instalar os pacotes. Clique com o botão direito do mouse no nó e escolha **Instalar do requirements.txt**.

Caso precise criar um ambiente virtual, consulte [Usar ambientes virtuais](#).

Gerar o arquivo requirements.txt

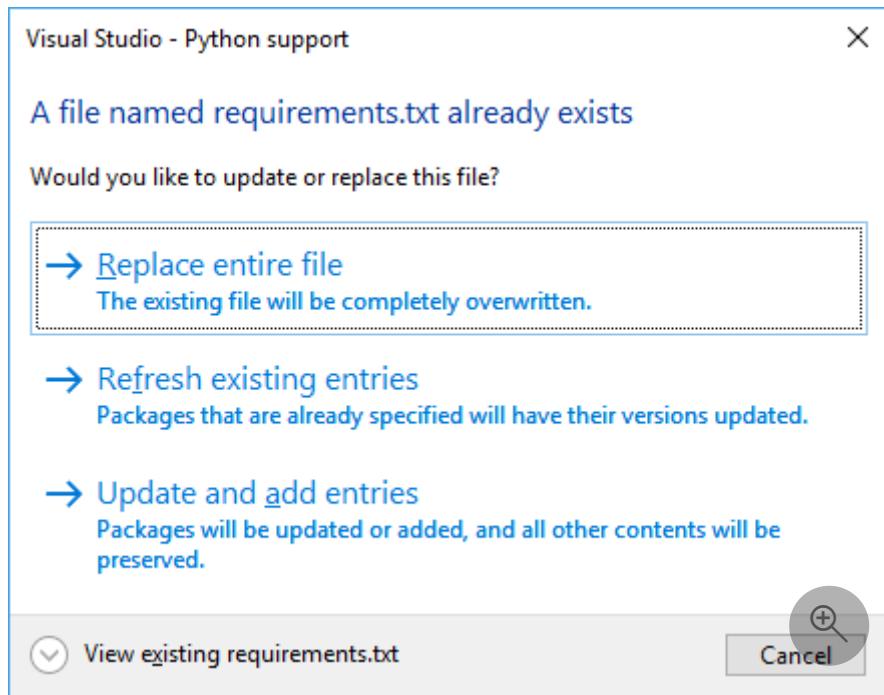
Se todos os pacotes do Python necessários ao projeto já estiverem instalados em um ambiente, você poderá gerar o arquivo `requirements.txt` no Visual Studio.

1. No **Gerenciador de Soluções**, expanda o projeto, depois expanda o nó **Ambientes do Python**.
2. Encontre o nó do ambiente para o qual deseja gerar o arquivo de requisitos. Clique com o botão direito do mouse no nó e escolha **Gerar o requirements.txt**.



Atualizar ou adicionar as entradas de um arquivo requirements.txt existente

Se o arquivo `requirements.txt` já existir, o Visual Studio exibirá uma solicitação com diversas opções:



- **Substituir o arquivo inteiro**: substitui todos os itens, comentários e opções definidos no arquivo `requirements.txt`.
- **Atualizar as entradas existentes**: atualiza os especificadores de versão no arquivo `requirements.txt` para que correspondam à versão instalada atualmente.
- **Atualizar e adicionar entradas**: atualiza os requisitos existentes no arquivo `requirements.txt` e acrescenta todos os novos requisitos de pacotes ao final do arquivo.

O Visual Studio executa o `pip` a fim de detectar os requisitos atuais de pacotes para o ambiente, depois atualiza o arquivo `requirements.txt` com base na sua seleção.

Instalar manualmente as dependências de pacotes

Se o PIP não instalar uma dependência de pacote definida no arquivo `requirements.txt`, toda a instalação falhará.

Há duas opções para resolver esse problema:

- Edite manualmente o arquivo `requirements.txt` a fim de excluir o pacote com falha, depois execute o processo de instalação novamente.
- Use as [opções de comando do PIP](#) para fazer referência a uma versão instalável do pacote.

Atualizar o arquivo de requisitos com pip wheel

Caso use o comando `pip wheel` para compilar uma dependência, você poderá adicionar a opção `--find-links <path>` ao arquivo `requirements.txt`.

1. Chame o comando `pip wheel` a fim de compilar a lista de dependências necessárias:

```
Console
```

```
pip wheel azure
```

A saída mostra os wheels compilados para os pacotes coletados:

```
Saída
```

```
Downloading/unpacking azure
  Running setup.py (path:C:\Project\env\build\azure\setup.py)
egg_info for package azure

Building wheels for collected packages: azure
  Running setup.py bdist_wheel for azure
    Destination directory: c:\project\wheelhouse
Successfully built azure
Cleaning up...
```

2. Acrescente as opções `find-links` e `no-index` com o requisito de versão do pacote ao arquivo `requirements.txt`:

```
Console
```

```
type requirements.txt
--find-links wheelhouse
--no-index
azure==0.8.0
```

3. Execute o processo de instalação do PIP com o arquivo de requisitos atualizado:

```
Console
```

```
pip install -r requirements.txt -v
```

A saída monitora o progresso do processo de instalação:

```
Saída
```

```
Downloading/unpacking azure==0.8.0 (from -r requirements.txt (line 3))
  Local files found: C:/Project/wheelhouse/azure-0.8.0-py3-none-
```

```
any.whl
Installing collected packages: azure
Successfully installed azure
Cleaning up...
Removing temporary dir C:\Project\env\build...
```

Conteúdo relacionado

- [Gerenciar ambientes do Python no Visual Studio](#)
 - [Selecionar um interpretador para um projeto](#)
 - [Caminhos de pesquisa](#)
 - [Referência da janela de ambientes do Python](#)
-

Comentários

Esta página foi útil?

 Yes

 No

Utilizar pastas do Python em caminhos de pesquisa do Visual Studio

Artigo • 18/04/2024

Em um programa típico do Python, a variável de ambiente `PYTHONPATH` (ou `IRONPYTHONPATH` etc.) fornece o caminho de pesquisa padrão para arquivos de módulo. As instruções `from <name> import...` or `import <name>` ordenam que o Python pesquise os locais específicos em busca de arquivos que correspondam à especificação de `<name>`. Os locais são pesquisados na seguinte ordem:

1. Módulos internos do Python
2. A pasta que contém o código do Python que está em execução.
3. O “caminho de pesquisa do módulo”, conforme definido pela variável de ambiente aplicável. Para obter mais informações, consulte as seções [The Module Search Path](#) e [Environment variables](#) na documentação básica do Python.

O Visual Studio ignora a variável de ambiente do caminho de pesquisa, mesmo quando ela tiver sido configurada para todo o sistema. Isso acontece porque a utilização da variável levanta questões que não são simples de responder, como:

- *Os módulos referenciados esperam uma instalação do Python 2.7, do Python 3.6 ou de outra versão?*
- *Os arquivos na variável de ambiente do caminho de pesquisa devem substituir os módulos de biblioteca padrão?*
- *Algum comportamento de substituição é esperado e abordado ou é possível que a ação seja maliciosa?*

Para auxiliar os desenvolvedores, o Visual Studio fornece uma forma de especificar caminhos de pesquisa diretamente nos projetos e ambientes do Python. O código que você executa ou depura no Visual Studio recebe os caminhos da variável de ambiente `PYTHONPATH` e da variável equivalente. Com a adição de caminhos de pesquisa, o Visual Studio inspeciona as bibliotecas nos locais especificados e cria bancos de dados do IntelliSense para as bibliotecas, conforme a necessidade. (No Visual Studio 2017 versão 15.5 e anteriores, a construção do banco de dados pode demorar um pouco, dependendo da quantidade de bibliotecas).

Pré-requisitos

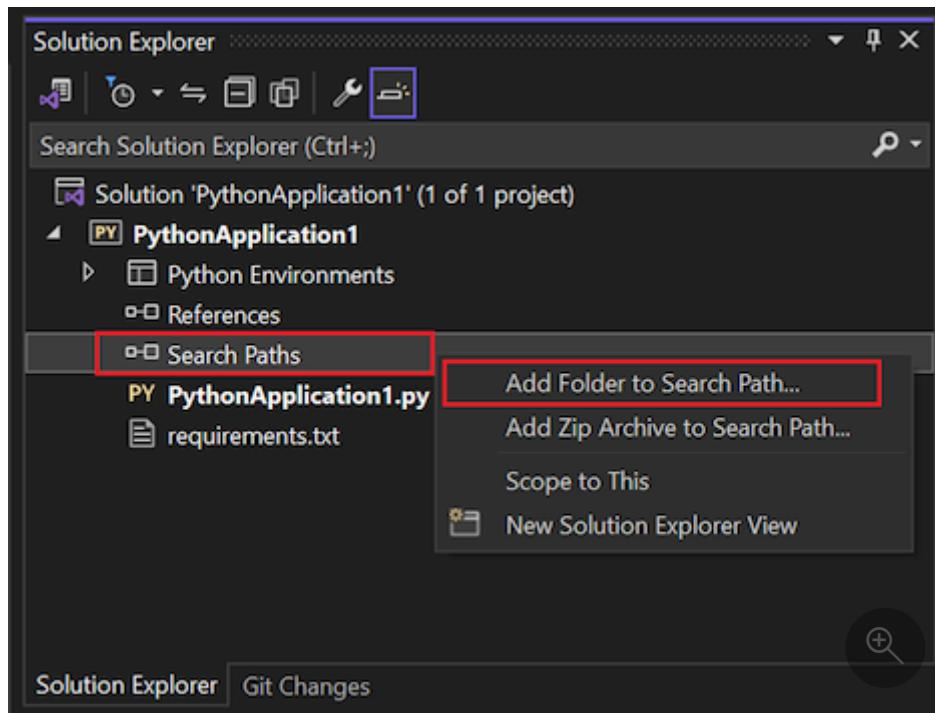
- Visual Studio. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).

- As pastas do Python que deseja adicionar aos caminhos de pesquisa.

Adicionar pastas do Python aos caminhos de pesquisa

Siga essas etapas para adicionar pastas do Python aos caminhos de pesquisa do Visual Studio:

1. No Gerenciador de Soluções, expanda o nó do projeto do Python, clique com o botão direito do mouse em **Caminhos de Pesquisa** e selecione **Adicionar Pasta ao Caminho de Pesquisa**:



2. Na caixa de diálogo, acesse o local da pasta que deseja adicionar aos caminhos de pesquisa reconhecidos.
3. Escolha a pasta e selecione **Selecionar pasta**.

Depois que as pastas forem adicionadas aos caminhos de pesquisa, o Visual Studio usará esses caminhos para qualquer ambiente associado ao projeto.

Observação

Se o seu ambiente for baseado no Python 3 e você tentar adicionar um caminho de pesquisa a módulos do Python 2.7, poderão ocorrer erros.

Adicionar arquivos zip e egg aos caminhos de pesquisa

É possível adicionar arquivos com extensão `.zip` ou `.egg` aos caminhos de pesquisa usando a opção **Adicionar Arquivo Zip ao Caminho de Pesquisa**. Assim como ocorre com pastas, o conteúdo desses arquivos é examinado e disponibilizado para o IntelliSense.

Conteúdo relacionado

- [Gerenciar ambientes do Python no Visual Studio](#)
- [Selecionar um interpretador para um projeto](#)
- [Use requirements.txt para dependências](#)
- [Referência da janela de ambientes do Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Referência as guias da janela Ambientes do Python

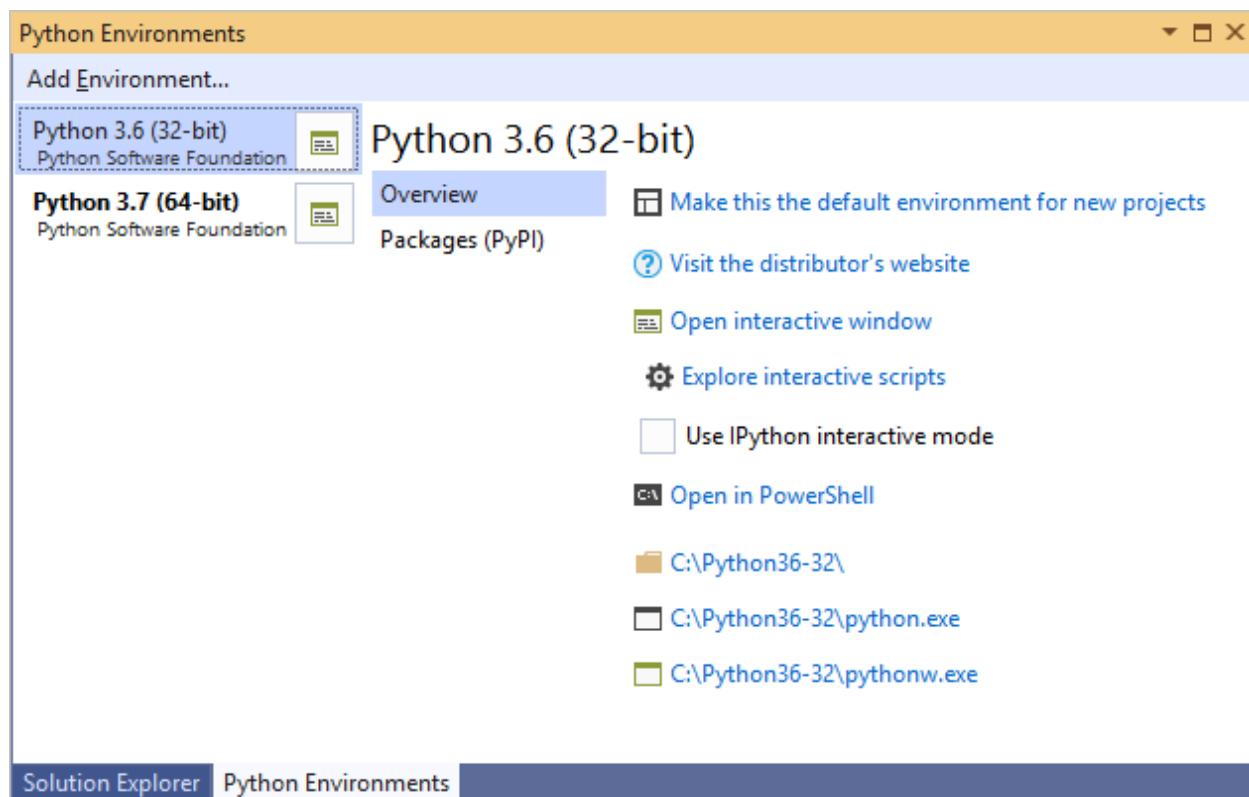
Artigo • 19/06/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Para abrir a janela Ambientes de Python:

- Selecione o comando de menu **Exibir>Outras Janelas>Ambientes do Python**.
- Clique com o botão direito do mouse no nó **Ambientes do Python** de um projeto no **Gerenciador de Soluções** e escolha **Exibir Todos os Ambientes do Python**.

Se você expandir a janela **Ambientes do Python** até um tamanho grande o suficiente, essas opções serão mostradas como guias, o que talvez você considere mais conveniente para trabalhar. Para maior clareza, as guias neste artigo são mostradas na exibição expandida.



Guia Visão Geral

Fornece informações básicas e comandos para o ambiente:

Python 3.6 (32-bit)

Overview

Make this the default environment for new projects

Packages (PyPI)

Visit the distributor's website

Open interactive window

Explore interactive scripts

Use IPython interactive mode

Open in PowerShell

C:\Python36-32\

C:\Python36-32\python.exe

C:\Python36-32\pythonw.exe

| Comando | Descrição |
|---|--|
| Tornar este ambiente padrão para novos projetos | Definir o ambiente ativo, o que pode fazer com que o Visual Studio (2017 versão 15.5 e anteriores) pare de responder brevemente enquanto carrega o banco de dados do IntelliSense. Ambientes com muitos pacotes podem parar de responder por um período maior. |
| Visitar o site do distribuidor | Abre um navegador para a URL fornecida para a distribuição de Python. Python 3.x, por exemplo, vai para python.org. |
| Abrir a janela interativa | Abre a janela interativa (REPL) para esse ambiente dentro do Visual Studio, aplicando quaisquer scripts de inicialização (veja abaixo). |
| Explorar scripts interativos | Veja os scripts de inicialização . |
| Usar o modo interativo do IPython | Quando definido, abre a janela Interativa com IPython por padrão. Isso habilita os gráficos embutidos e a sintaxe estendida do IPython como <code>name?</code> para exibir a ajuda e <code>!command</code> para comandos shell. Essa opção é recomendada quando estiver usando uma distribuição Anaconda, pois ela requer pacotes extras. Para obter mais informações, confira Usar o IPython na Janela Interativa . |
| Abrir no PowerShell | Inicia o interpretador em uma janela de comando do PowerShell. |
| (Links de pasta e do programa) | Os interpretadores <code>python.exe</code> e <code>pythonw.exe</code> fornecem acesso rápido à pasta de instalação do ambiente. O primeiro abre no Windows Explorer, os dois últimos abrem uma janela do console. |

Scripts de inicialização

Como você janelas interativas no fluxo de trabalho diário, provavelmente desenvolverá funções auxiliares que você usa regularmente. Por exemplo, você pode criar uma função que abre um DataFrame no Excel e, em seguida, salva esse código como um script de inicialização para que ele esteja sempre disponível na janela **Interativa**.

Os scripts de inicialização contêm o código que a janela **Interativa** carrega e executa automaticamente, incluindo importações, definições de função e literalmente qualquer outra coisa. Esses scripts são referenciados de duas maneiras:

1. Quando você instala um ambiente, o Visual Studio cria uma pasta *Documents\Visual Studio <versão>\Python Scripts\<ambiente>*, em que <versão> é a versão do Visual Studio (como 2017 ou 2019) e <ambiente> corresponde ao nome do ambiente. Você pode navegar facilmente para a pasta específica do ambiente com o comando **Explorar scripts interativos**. Quando você inicia a janela **Interativa** para esse ambiente, ela carrega e executa qualquer arquivo .py que for encontrado aqui em ordem alfabética.
2. O controle **Scripts** na guia **Ferramentas>Opções>Python>Janelas Interativas** (confira [Opções de janelas Interativas](#)) destina-se a especificar uma pasta adicional para os scripts de inicialização que estão carregados e são executados em todos os ambientes. No entanto, esse recurso não funciona no momento.

Guia Configurar

Se estiver disponível, a guia **Configurar** conterá detalhes, conforme descrito na tabela abaixo. Se essa guia não estiver presente, isso significa que o Visual Studio está gerenciando todos os detalhes automaticamente.

python36-32

| | | |
|---------------------------------|---|-----------------------------|
| Overview | Description | Apply |
| Configure | python36-32 | Reset |
| Packages (PyPI) | Prefix path | Auto Detect |
| | c:\python36-32 | Remove |
| | ... | |
| | Interpreter path | |
| | c:\python36-32\python.exe | ... |
| | ... | |
| | Windowed interpreter | |
| | c:\python36-32\pythonw.exe | ... |
| | ... | |
| | Language version | |
| | 3.6 | ▼ |
| | | |
| | Architecture | |
| | 32-bit | ▼ |
| | | |
| | Path environment variable | |
| | PYTHONPATH | |

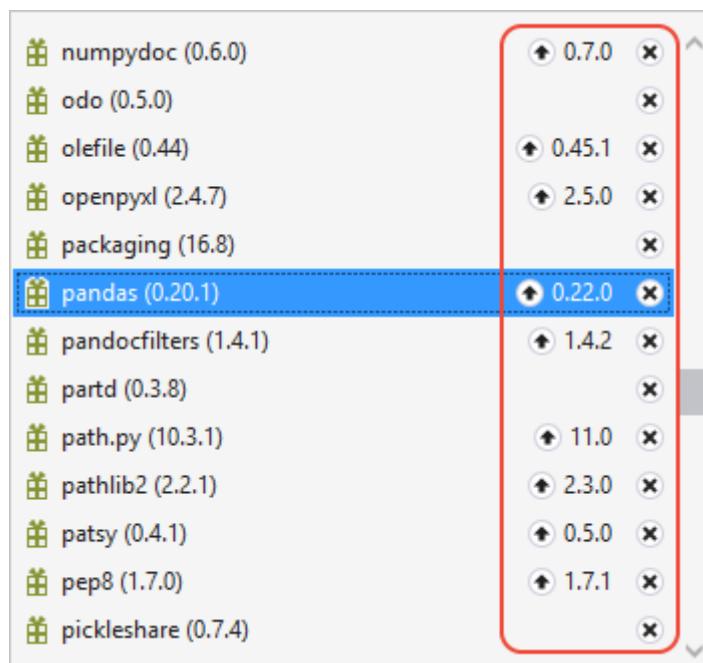
| Campo | Descrição |
|---|---|
| Descrição | O nome a ser fornecido para o ambiente. |
| Caminho do prefixo | A localização da pasta base do interpretador. Ao preencher esse valor e clicar em Detecção Automática , o Visual Studio tenta preencher os outros campos para você. |
| Caminho do interpretador | O caminho para o executável do interpretador, normalmente, o caminho do prefixo seguido por python.exe |
| Interpretador em janelas | O caminho para o executável que não é de console, geralmente, é o caminho do prefixo seguido por pythonw.exe . |
| Caminho da biblioteca (se estiver disponível) | Especifica a raiz da biblioteca padrão, mas esse valor poderá ser ignorado se o Visual Studio conseguir solicitar um caminho mais preciso do interpretador. |
| Versão da linguagem | Selecionada no menu suspenso. |
| Arquitetura | Normalmente, detectada e preenchida automaticamente. Caso contrário, especifica 32 bits ou 64 bits . |
| Variável de ambiente do caminho | A variável de ambiente que o interpretador usa para encontrar caminhos de pesquisa. O Visual Studio altera o valor da variável ao iniciar o Python, para que ela contenha os caminhos de pesquisa do projeto. Normalmente, essa propriedade deve ser definida como PYTHONPATH , mas alguns interpretadores usam outro valor. |

Guia Pacotes

Também chamada de "pip" em versões anteriores.

Gerencia os pacotes instalados no ambiente usando pip (a guia **Pacotes (PyPI)**) ou o conda (a guia **Pacotes (Conda)**, para ambientes do conda no Visual Studio 2017 versão 15.7 e posteriores). Nessa guia, você também pode procurar e instalar novos pacotes, incluindo as dependências dele.

Os pacotes que já estão instalados são exibidos com controles para atualizar (uma seta para cima) e desinstalar (X em um círculo) o pacote:



Inserir um termo de pesquisa filtra a lista de pacotes instalados, bem como os pacotes que podem ser instalados do PyPI.

Python 3.6 (32-bit)

Overview

Packages (PyPI)

num

- numpy (1.15.0) 1.16.2
- selenium (3.14.1) 3.141.0

Run command: pip install num

Install num2chars

Install num2cyrillic

Install num2es

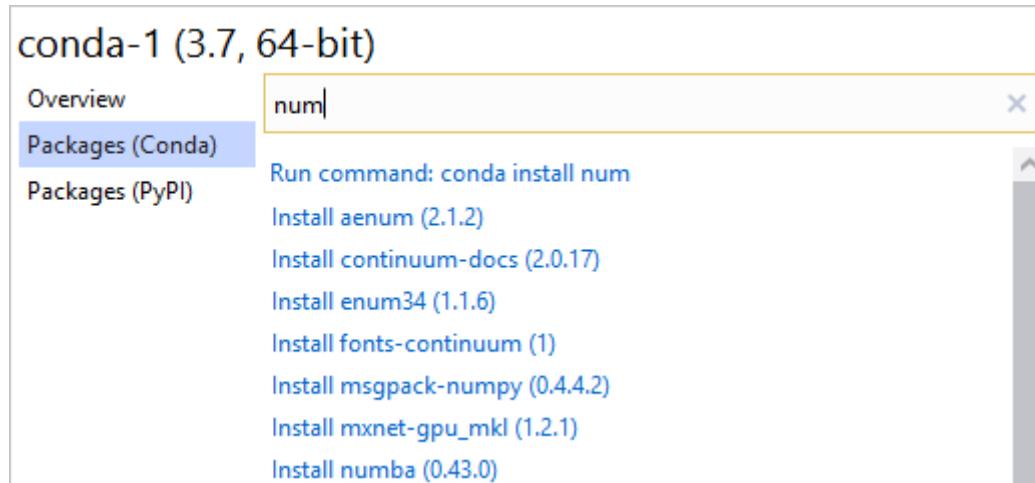
Install num2fawords

Install num2str

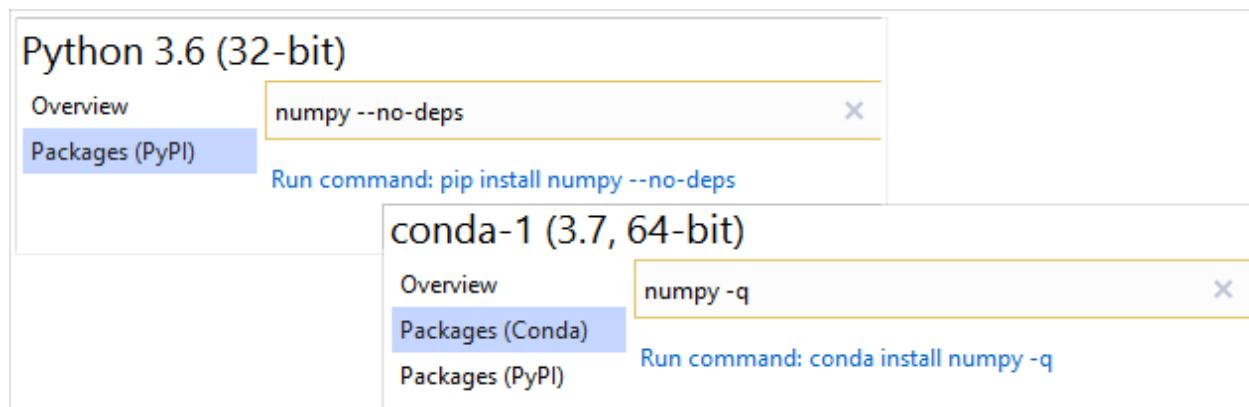
Install num2tex

Install num2words

Como você pode ver na imagem acima, os resultados da pesquisa mostram vários pacotes que correspondem ao termo da pesquisa. A primeira entrada na lista, no entanto, é um comando para executar `pip install <name>` diretamente. Caso esteja na guia Pacotes (Conda), você verá `conda install <name>`:



Em ambos os casos, você pode personalizar a instalação pela adição de argumentos na caixa de pesquisa após o nome do pacote. Quando você inclui argumentos, os resultados da pesquisa mostram `pip install` ou `conda install` seguido do conteúdo da caixa de pesquisa:

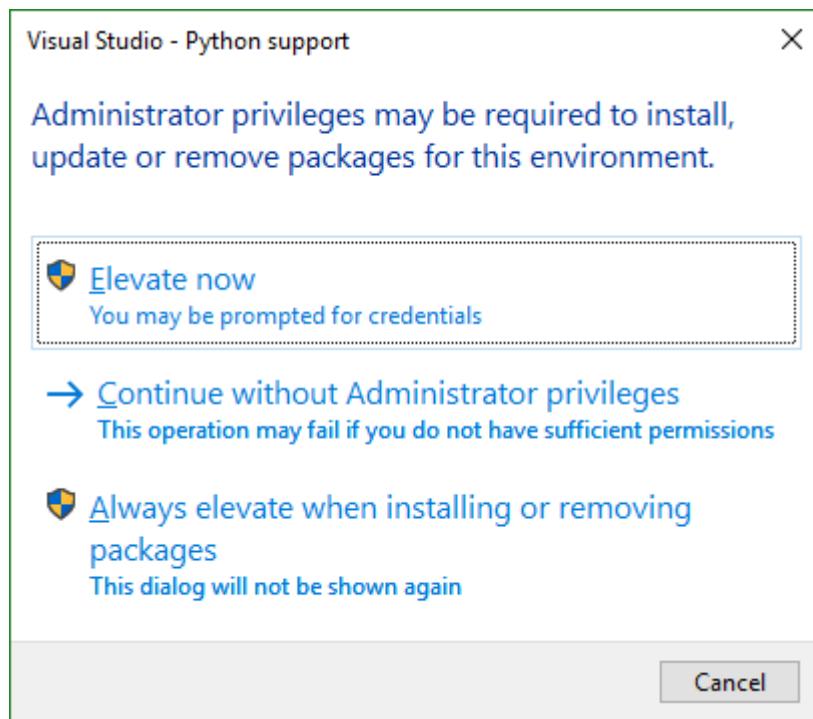


Instalar um pacote cria subpastas dentro da pasta *Lib* do ambiente no sistema de arquivos. Por exemplo, se você tiver Python 3.6 instalado em *c:\Python36*, os pacotes são instalados em *c:\Python36\Lib*, se você tiver o Anaconda3 instalado em *c:\Program Files\Anaconda3*, os pacotes serão instalados em *c:\Program Files\Anaconda3\Lib*. Nos ambientes do conda, os pacotes são instalados na pasta do ambiente.

Conceder privilégios de administrador à instalação do pacote

Ao instalar os pacotes em um ambiente que está localizado em uma área protegida do sistema de arquivos, como *c:\Program Files\Anaconda3\Lib*, o Visual Studio deve executar `pip install` com privilégios elevados para permitir que ele crie subpastas do

pacote. Quando a elevação é necessária, o Visual Studio exibe o prompt **Podem ser necessários privilégios de administrador para instalar, atualizar ou remover pacotes para esse ambiente:**



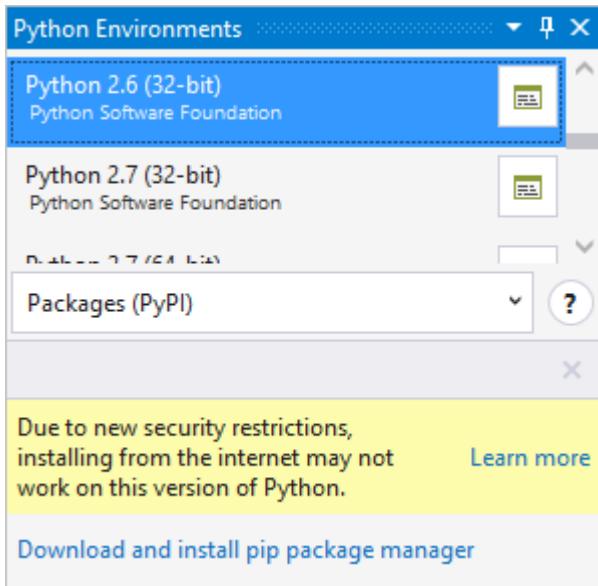
Elevar agora concede privilégios administrativos para executar o PIP para uma única operação, sujeita também a qualquer prompt de permissão do sistema operacional. Escolher **Continuar sem privilégios de administrador** tenta instalar o pacote, mas o PIP falha ao tentar criar pastas com uma saída como erro: **não foi possível criar 'C:\Arquivos de Programas\Anaconda3\Lib\site-packages\png.py': permissão negada.**

Selecionar **Sempre elevar ao instalar ou remover pacotes** impede que a caixa de diálogo apareça para o ambiente em questão. Para fazer a caixa de diálogo aparecer novamente, vá para **Ferramentas > Opções > Python > Geral** e escolha o botão **Redefinir todas as caixas de diálogo permanentemente ocultas**.

Nessa mesma guia de **Opções**, você também pode escolher **Sempre executar o PIP como administrador** para suprimir a caixa de diálogo para todos os ambientes. Consulte [Opções – guia Geral](#).

Restrições de segurança com versões mais antigas do Python

Ao usar o Python 2.6, 3.1 e 3.2, o Visual Studio mostra o aviso **Devido a restrições de segurança, a instalação por meio da Internet pode não funcionar nesta versão do Python:**



O motivo para o aviso é que, com essas versões mais antigas do Python, `pip install` não dá suporte para o protocolo TLS 1.2, que é necessário para baixar pacotes da origem do pacote, pypi.org. Builds personalizados do Python talvez sejam compatíveis com TLS 1.2 e, nesse caso, `pip install` poderá funcionar.

É possível baixar o *get-pip.py* apropriado para um pacote em bootstrap.pypa.io, fazer o download manual de um pacote em pypi.org e, em seguida, instalar o pacote dessa cópia local.

No entanto, a recomendação é apenas atualizar para uma versão recente do Python; nesse caso, o aviso não é exibido.

Confira também

- [Gerenciar ambientes do Python no Visual Studio](#)
- [Selecionar um interpretador para um projeto](#)
- [Usar requirements.txt para dependências](#)
- [Caminhos de pesquisa](#)

Configurar aplicativos Web do Python para o IIS

Artigo • 18/04/2024

Ao usar os Serviços de Informações da Internet (IIS) como um servidor Web em um computador Windows (incluindo [máquinas virtuais do Windows no Azure](#)), você precisa configurar o aplicativo Web Python para permitir que o IIS processe corretamente o código Python. A configuração é realizada por meio de ajustes no arquivo `web.config` para o aplicativo Web em Python. Este artigo descreve como definir as configurações necessárias.

Pré-requisitos

- Python no Windows instalado. Para executar um aplicativo Web, primeiro instale a versão necessária do Python diretamente no computador host do Windows, conforme descrito em [Instalar interpretadores do Python](#).
 - Identificar o local do interpretador `python.exe`. Para sua conveniência, você pode adicionar esse local à variável de ambiente PATH.
- Pacotes necessários instalados. Para um host dedicado, você pode usar o ambiente global do Python para executar o aplicativo em vez de um ambiente virtual. Da mesma forma, você pode instalar todos os requisitos do aplicativo no ambiente global executando o comando `pip install -r requirements.txt`.

Configurar o `web.config` para apontar para o interpretador do Python

O arquivo `web.config` do aplicativo em Python instrui o servidor Web do IIS (versão 7 ou posterior) em execução no Windows sobre como ele deve tratar as solicitações do Python por meio de `HttpPlatformHandler` (recomendado) ou `FastCGI`. As versões do Visual Studio 2015 e anterior fazem essas modificações automaticamente. No Visual Studio 2017 e posterior, você deve modificar o arquivo `web.config` manualmente.

Se o projeto ainda não contiver um arquivo `web.config`, você pode adicionar um clicando com o botão direito do mouse no diretório do projeto, selecionando **Adicionar > Novo Item** e procurando por `web.config` ou criando um arquivo XML `web.config` em branco.

Configurar o HttpPlatformHandler

O módulo `HttpPlatform` passa conexões de soquete diretamente para um processo de Python autônomo. Essa passagem permite que você execute qualquer servidor Web que desejar, mas ela requer um script de inicialização que executa um servidor Web local. Essa abordagem é comumente feita usando um framework web Python, como Flask ou Django. Você especifica o script no elemento `<httpPlatform>` do arquivo `web.config`. O atributo `processPath` aponta para o interpretador Python da extensão do site. O atributo `arguments` aponta para o script de inicialização que executa um servidor Web local, neste caso, `runserver.py`, e quaisquer argumentos que você deseja fornecer:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.webServer>
        <handlers>
            <add name="PythonHandler" path="*" verb="*"
modules="httpPlatformHandler" resourceType="Unspecified"/>
        </handlers>
        <httpPlatform processPath="c:\python36-32\python.exe"
            arguments="c:\home\site\wwwroot\runserver.py --port
%HTTP_PLATFORM_PORT%"
            stdoutLogEnabled="true"
            stdoutLogFile="c:\home\LogFiles\python.log"
            startupTimeLimit="60"
            processesPerApplication="16">
            <environmentVariables>
                <environmentVariable name="SERVER_PORT" value="%HTTP_PLATFORM_PORT%">
            />
            </environmentVariables>
        </httpPlatform>
    </system.webServer>
</configuration>
```

Neste exemplo, a variável de ambiente `HTTP_PLATFORM_PORT` contém a porta na qual o servidor local deve escutar as conexões do `localhost`. Este exemplo também mostra como criar outra variável de ambiente, `SERVER_PORT`. Você pode criar e atribuir variáveis de ambiente conforme necessário.

Configurar o manipulador do FastCGI

O FastCGI é uma interface que funciona no nível da solicitação. O IIS recebe conexões de entrada e encaminha cada solicitação para um aplicativo WSGI em execução em um ou mais processos Python persistentes.

ⓘ Observação

É recomendável usar **HttpPlatformHandler** para configurar os aplicativos, pois o projeto [WFastCGI](#) deixou de receber manutenção.

Para usar o FastCGI, primeiro instale e configure o pacote wfastcgi, conforme descrito em pypi.org/project/wfastcgi/.

Em seguida, modifique o arquivo `web.config` do aplicativo para incluir os caminhos completos para o executável `python.exe` e o arquivo `wfastcgi.py` na chave `PythonHandler`. As etapas a seguir pressupõem que o Python esteja instalado na pasta `c:\python36-32` e o código do aplicativo esteja na pasta `c:\home\site\wwwroot`. Ajuste esses valores para seus caminhos corretamente.

1. Modifique a entrada `PythonHandler` no arquivo `web.config` para que o caminho corresponda ao local de instalação do Python. Para obter mais informações, confira [Referência de configuração do IIS](#) (iis.net).

XML

```
<system.webServer>
  <handlers>
    <add name="PythonHandler" path="*" verb="*"
modules="FastCgiModule"
      scriptProcessor="c:\python36-32\python.exe|c:\python36-
32\wfastcgi.py"
      resourceType="Unspecified" requireAccess="Script"/>
  </handlers>
</system.webServer>
```

2. Na seção `<appSettings>` do arquivo `web.config`, adicione chaves para `WSGI_HANDLER`, `WSGI_LOG` (opcional) e `PYTHONPATH`:

XML

```
<appSettings>
  <add key="PYTHONPATH" value="c:\home\site\wwwroot"/>
  <!-- The handler here is specific to Bottle; see the next section. --
->
  <add key="WSGI_HANDLER" value="app.wsgi_app()"/>
  <add key="WSGI_LOG" value="c:\home\LogFiles\wfastcgi.log"/>
</appSettings>
```

Esses valores `<appSettings>` estão disponíveis para seu aplicativo como variáveis de ambiente:

- O valor da chave `PYTHONPATH` pode ser estendido livremente, mas deve incluir a raiz do aplicativo.
- A chave `WSGI_HANDLER` deve apontar para um aplicativo WSGI importável do seu aplicativo.
- A chave `WSGI_LOG` é opcional, mas é recomendada para a depuração do aplicativo.

3. Defina a entrada `WSGI_HANDLER` nos arquivos `web.config` de acordo com a estrutura que você está usando:

- **Bottle:** inclua parênteses depois do valor `app.wsgi_app` conforme mostrado no exemplo. Os parênteses são necessários porque o objeto é uma função, não uma variável. Você pode ver a sintaxe no arquivo `app.py`.

XML

```
<!-- Bottle apps only -->
<add key="WSGI_HANDLER" value="app.wsgi_app()" />
```

- **Flask:** altere o valor de `WSGI_HANDLER` para `<project_name>.app`, em que `<project_name>` corresponde ao nome do seu projeto. Você pode localizar o identificador exato examinando a instrução `from <project_name> import app` no arquivo `runserver.py`. Por exemplo, se o projeto fosse denominado `FlaskAzurePublishExample`, a entrada seria semelhante ao seguinte:

XML

```
<!-- Flask apps only: Change the project name to match your app --
>
<add key="WSGI_HANDLER" value="FlaskAzurePublishExample.app" />
```

- **Django:** duas alterações são necessárias no arquivo `web.config` para projetos do Django.
 - Altere o valor de `WSGI_HANDLER` para `django.core.wsgi.get_wsgi_application()`. O objeto está no arquivo `wsgi.py`.

XML

```
<!-- Django apps only -->
<add key="WSGI_HANDLER"
value="django.core.wsgi.get_wsgi_application()" />
```

- Adicione a seguinte entrada imediatamente após a entrada da chave `WSGI_HANDLER`. Substitua o valor `DjangoAzurePublishExample` pelo nome do projeto:

XML

```
<add key="DJANGO_SETTINGS_MODULE"  
      value="django_iis_example.settings" />
```

4. Somente aplicativos Django: no arquivo `settings.py` do projeto do Django, adicione o domínio de URL do site ou o endereço IP à entrada `ALLOWED_HOSTS`. Substitua "1.2.3.4" pelo seu URL ou endereço IP:

Python

```
# Change the URL or IP address to your specific site  
ALLOWED_HOSTS = ['1.2.3.4']
```

Se você não adicionar o URL aos resultados da matriz, verá o seguinte erro:

Saída

```
DisallowedHost at / Invalid HTTP_HOST header: '\<site URL\>'. You might  
need to add '\<site URL\>' to ALLOWED_HOSTS.
```

Quando a matriz está vazia, o Django permite automaticamente `'localhost'` e `'127.0.0.1'` como hosts. Se você adicionar o URL de produção, esses sites de host não serão permitidos automaticamente. Por esse motivo, convém manter cópias do arquivo `settings.py` de desenvolvimento e de produção separadas ou usar variáveis de ambiente para controlar os valores de runtime.

Implantar IIS ou uma máquina virtual do Windows

Quando tiver o arquivo `web.config` correto no projeto, você pode publicar no computador que está executando o IIS do **Gerenciador de Soluções**. Clique com o botão direito do mouse no projeto, selecione **Publicar** e, em seguida, selecione **IIS, FTP etc.** Nessa situação, o Visual Studio copiará somente os arquivos de projeto para o servidor. Você é responsável por toda a configuração do lado do servidor.

Conteúdo relacionado

- Referência de configuração do IIS [\(iis.net\)](#)
 - Instalar interpretadores do Python
 - Máquinas virtuais do Windows no Azure
-

Comentários

Esta página foi útil?

 Yes

 No

Editar o código Python e usar o IntelliSense

Artigo • 18/04/2024

Como você passa muito tempo dedicado ao desenvolvimento no editor de códigos, o [Suporte para Python no Visual Studio](#) fornece funcionalidade para ajudá-lo a ser mais produtivo. Os recursos incluem o realce de sintaxe do IntelliSense, o preenchimento automático, a ajuda de assinatura, as substituições de método, a pesquisa e a navegação.

O editor de códigos é integrado à janela **Interativa** no Visual Studio. À medida que você trabalha, a troca de código entre as duas janelas é simples. Para obter mais informações, consulte [Etapa 3 do tutorial: Usar a janela do REPL Interativo](#) e [Usar a janela Interativa – comando Enviar para Interativa](#).

A [Estrutura de tópicos](#) ajuda você a manter o foco em seções específicas do código. Para obter uma documentação geral sobre edição do código no Visual Studio, confira [Recursos do editor de código](#).

O **Pesquisador de Objetos** do Visual Studio possibilita que você inspecione as classes do Python definidas em cada módulo e as funções definidas nessas classes. Acesse esse recurso pelo menu **Exibir** ou utilizando o atalho do teclado **Ctrl+Alt+J**.

Usar os recursos do IntelliSense

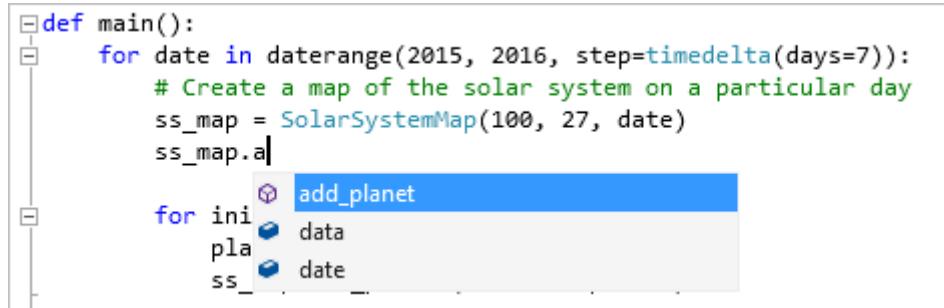
O IntelliSense fornece [preenchimentos](#), [ajuda da assinatura](#), [informações rápidas](#) e [coloração de código](#). O Visual Studio 2017 versão 15.7 e posteriores também dão suporte a [dicas de tipo](#).

Para melhorar o desempenho, o IntelliSense no Visual Studio 2017 versão 15.5 e em versões anteriores depende do banco de dados de preenchimento que é gerado para cada ambiente do Python no projeto. Poderá ser necessário atualizar o banco de dados ao adicionar, remover ou atualizar pacotes. O status do banco de dados é mostrado na janela **Ambientes do Python** (um complemento do **Gerenciador de Soluções**) na guia **IntelliSense**. Para obter mais informações, consulte a [Referência da janela Ambientes](#).

O Visual Studio 2017 versão 15.6 e posterior usa um modo diferente para fornecer as conclusões de IntelliSense que não são dependentes do banco de dados.

Preenchimentos

Preenchimentos aparecem como instruções, identificadores e outras palavras que podem ser inseridas adequadamente no local atual no editor. O IntelliSense preenche a lista de opções com base no contexto e filtra itens incorretos ou confusos. Em geral, os preenchimentos são acionados pela inserção de diferentes instruções (como `import`) e operadores (incluindo um ponto final), mas eles podem aparecer a qualquer momento por meio da seleção do atalho do teclado **Ctrl+J + Espaço**.



A screenshot of an IDE showing code completion. The code is:

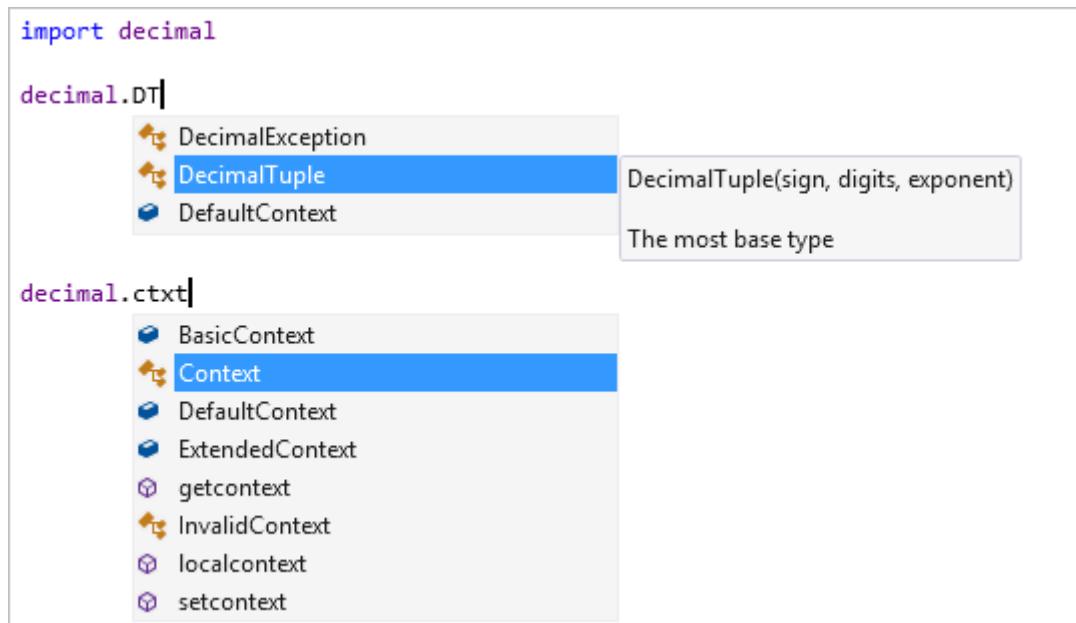
```
def main():
    for date in daterange(2015, 2016, step=timedelta(days=7)):
        # Create a map of the solar system on a particular day
        ss_map = SolarSystemMap(100, 27, date)
        ss_map.a|
```

The cursor is at the end of `ss_map.a`. A completion dropdown shows several options starting with 'a': `add_planet`, `ini`, `pla`, and `ss`. The option `add_planet` is highlighted.

Quando uma lista de preenchimento é aberta, é possível pesquisar o preenchimento que você deseja usando as teclas de direção e o mouse ou continuando a digitação. Conforme você digita mais letras, a lista é filtrada ainda mais para mostrar os prováveis preenchimentos. Você também pode usar atalhos, como:

- Digitar letras que não estão no início do nome, como "parse" para encontrar "argparse".
- Digitar apenas letras que estão no início de palavras, como "abc" para encontrar "AbstractBaseClass" ou "air" para encontrar "as_integer_ratio".
- Ignorar letras, como "b64" para encontrar "base64".

Estes são alguns exemplos:



A screenshot of an IDE showing code completion for the `decimal` module. The code is:

```
import decimal

decimal.DT|
```

The cursor is at the end of `decimal.DT`. A completion dropdown shows three options: `DecimalException`, `DecimalTuple` (which is highlighted), and `DefaultContext`. To the right of the dropdown, a tooltip says: "The most base type".


```
decimal.ctx|
```

The cursor is at the end of `decimal.ctx`. A completion dropdown shows several options: `BaseContext`, `Context` (which is highlighted), `DefaultContext`, `ExtendedContext`, `getcontext`, `InvalidContext`, `localcontext`, and `setcontext`.

Os preenchimentos de membro aparecem automaticamente quando você digita um ponto final depois de uma variável ou um valor, juntamente com os métodos e os

atributos dos tipos possíveis. Se for possível que uma variável seja de mais de um tipo, a lista incluirá todas as possibilidades de todos os tipos. Informações adicionais são mostradas para indicar quais tipos oferecem suporte a cada conclusão. Quando todos os tipos possíveis oferecem suporte a uma conclusão, nenhuma anotação é exibida.

```
value = ['a', 'list']
value = 'a', 'tuple'
value = 3.1415

value.|
```

The screenshot shows a code editor with the following code:

```
value = ['a', 'list']
value = 'a', 'tuple'
value = 3.1415

value.|
```

A completion dropdown is open at the cursor position. It lists several methods for the current context:

- append (list)
- as_integer_ratio (float)
- clear (list)
- conjugate (float)
- copy (list)
- count (tuple, list)
- extend (list)
- fromhex (float)
- hex (float)

Por padrão, os membros “dunder” (membros que começam e terminam com um sublinhado duplo) não são exibidos. Em geral, esses membros não devem ser acessados diretamente. Se você precisar usar um dunder, digite o sublinhado duplo à esquerda para adicionar esses preenchimentos à lista:

```
value = 3.1415

value._|
```

The screenshot shows a code editor with the following code:

```
value = 3.1415

value._|
```

A completion dropdown is open at the cursor position. It lists several dunder methods:

- __abs__
- __add__
- __bool__
- __class__
- __delattr__
- __dir__
- __divmod__
- __doc__
- __eq__

As instruções `import` e `from ... import` exibem uma lista de módulos que podem ser importados. A instrução `from ... import` produz uma lista que inclui os membros que podem ser importados do módulo especificado.

The screenshot shows the Visual Studio code editor with the following code:

```
import decimal
from decimal import Decimal
```

Two dropdown menus are open, showing type hints:

- The first dropdown for `decimal` contains: `_codecs_iso2022`, `_decimal`, and `_pydecimal`.
- The second dropdown for `Decimal` contains: `Decimal`, `DecimalException`, and `DecimalTuple`.

As instruções `raise` e `except` exibem listas de classes que provavelmente são tipos de erros. A lista pode não incluir todas as exceções definidas pelo usuário, mas ela ajuda você a encontrar as exceções internas adequadas rapidamente:

The screenshot shows the Visual Studio code editor with the following code:

```
try:
    raise N
except (S:
    pass
```

Two dropdown menus are open, showing type hints:

- The first dropdown for `N` contains: `FileNotFoundException`, `NameError`, `NotADirectoryError`, and `NotImplementedError`.
- The second dropdown for `S` contains: `OSSError`, `StopIteration`, `SyntaxError`, `SystemError`, and `SystemExit`.

A seleção do símbolo @ (arroba) inicializa um decorador e exibe outros decoradores potenciais. Muitos desses itens não podem ser utilizados como decoradores. Consulte a documentação da biblioteca para estabelecer qual decorador utilizar.

The screenshot shows the Visual Studio code editor with the following code:

```
1 class MyClass:
2     @
3         d classmethod
4             compile
5                 complex
6                     ConnectionAbortedError
```

A tooltip for `classmethod` is displayed:

- `classmethod(function) -> method`
- `Convert a function to be a class method.`
- `A class method receives the class as implicit first argument,`

Para obter mais informações, consulte [Opções: resultados de conclusão](#).

Dicas de tipo

As dicas de tipo estão disponíveis no Visual Studio 2017 versão 15.7 e em versões posteriores.

As "dicas de tipo" no Python 3.5+ ([PEP 484](#)) ([python.org](#)) são uma sintaxe de anotação para funções e classes que indicam os tipos de argumentos, valores de retorno e atributos de classe. O IntelliSense exibe dicas de tipo quando você focaliza argumentos, variáveis e chamadas de função que contêm essas anotações.

No exemplo a seguir, a classe `Vector` é declarada como o tipo `List[float]` e a função `scale` contém dicas de tipo para seus argumentos e o valor retornado. Passar o mouse sobre uma chamada da função mostra as dicas de tipo:

```
from typing import List
Vector = List[float]

def scale(scalar: float, vector: Vector) -> List:
    return [scalar * num for num in vector]

new_vector = scale(2.0, [1.0, -4.2, 5.4])
    scale: def app.scale(scalar: float, vector: list, list[float]) -> list
```

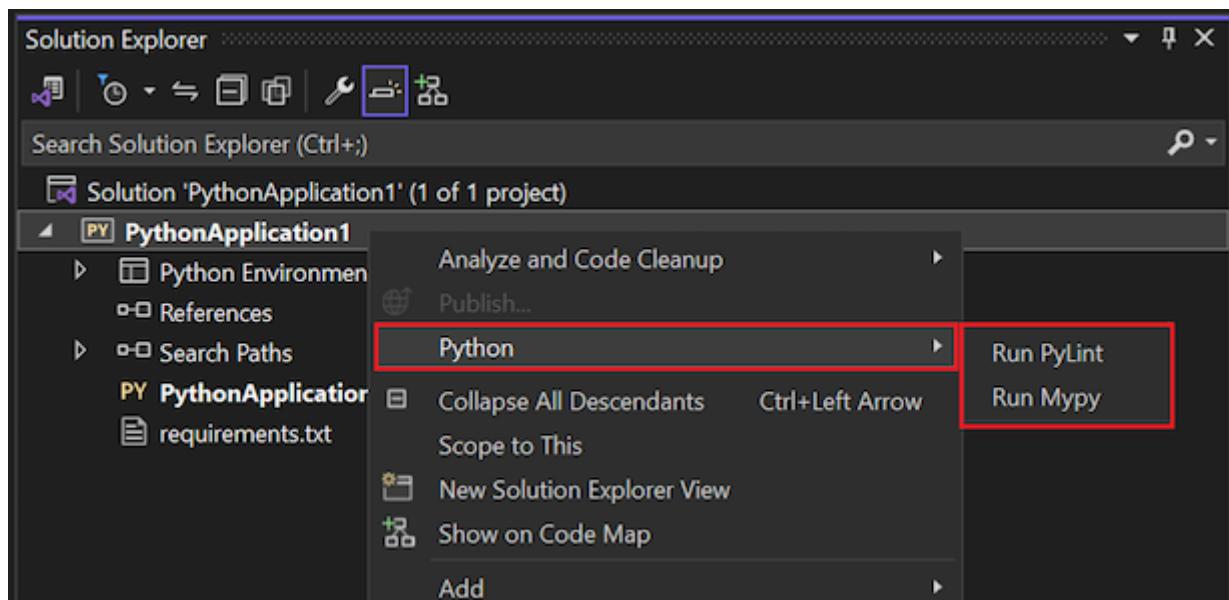
No próximo exemplo, você pode ver como os atributos anotados da classe `Employee` aparecem no pop-up de conclusão de IntelliSense para um atributo:

```
from typing import List
Vector = List[float]

def scale(scalar: float, vector: Vector) -> List:
    return [scalar * num for num in vector]

new_vector = scale(2.0, [1.0, -4.2, 5.4])
    scale: def app.scale(scalar: float, vector: list, list[float]) -> list
```

Também é útil validar as dicas de tipo em todo o seu projeto, pois erros normalmente não aparecem até o tempo de execução. Para isso, o Visual Studio integra a ferramenta padrão do setor MyPy usando o comando de menu de contexto **Python>Executar Mypy no Gerenciador de Soluções**:



A execução do comando solicitará que você instale o pacote do MyPy, se necessário. Depois, o Visual Studio executará o MyPy para validar as dicas de tipo em todos os arquivos Python do projeto. Os erros aparecem na janela **Lista de Erros** do Visual Studio. Selecionar um item na janela navega para a linha apropriada no seu código.

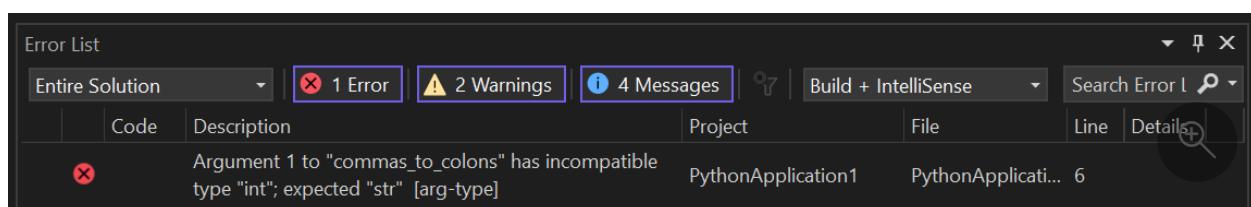
Como um exemplo simples, a definição de função a seguir contém uma dica de tipo que indica que o argumento `input` é do tipo `str`, enquanto a chamada para essa função tenta passar um número inteiro:

```
Python

def commas_to_cols(input: str):
    items = input.split(',')
    items = [x.strip() for x in items]
    return ':' .join(items)

commas_to_cols(1)
```

Usar o comando **Execute Mypy** neste código gera o seguinte erro:



⚠ Observação

Para as versões do Python anteriores à 3.5, o Visual Studio também exibe dicas de tipo fornecidas por meio de *arquivos stub* do Typeshed (`.pyi`). Você pode usar arquivos stub quando não quiser incluir dicas de tipo diretamente no código ou

para criar dicas de tipo para uma biblioteca que não as usa diretamente. Para obter mais informações, confira [Criar stubs para módulos do Python](#) no wiki de projeto do MyPy.

O Visual Studio ainda não oferece suporte a dicas de tipo nos comentários.

Ajuda da assinatura

Quando você escreve código que chama uma função, a ajuda de assinatura é exibido ao digitar o primeiro parêntese (. Ela apresenta toda a documentação disponível e informações sobre os parâmetros. Acesse a ajuda de assinatura pelo atalho do teclado Ctrl+Shift+Space dentro de uma chamada de função. As informações exibidas dependem das cadeias de caracteres de documentação no código-fonte da função, mas incluem os valores padrão.

```
import decimal

decimal.Decimal()
Decimal(value: str = '0', context: Context = None)
Create a decimal point instance.
...
```

💡 Dica

Para desabilitar a ajuda de assinatura, acesse **Ferramentas>Opções>Editor de Texto>Python>Geral**. Desmarque a caixa de seleção **Conclusão da instrução>Informações sobre parâmetros**.

Informações rápidas

Focalizar o ponteiro do mouse em um identificador exibe uma dica de ferramenta Informações Rápidas. Dependendo do identificador, as Informações Rápidas poderão exibir os possíveis valores ou tipos, toda a documentação disponível, os tipos de retorno e os locais de definição:

```
from decimal import *
d = Decimal(123)
d: Decimal instance
d.conjugate
d.conjugate: method conjugate of Decimal objects -> Decimal instance
```

Coloração de código

A coloração de código usa informações da análise de código para colorir variáveis, instruções e outras partes do código. As variáveis relacionadas aos módulos ou às classes poderão ser exibidas em uma cor diferente das funções ou dos demais valores. Os nomes de parâmetro poderão ser exibidos em uma cor diferente das variáveis locais ou globais. Por padrão, as funções não são exibidas em negrito.

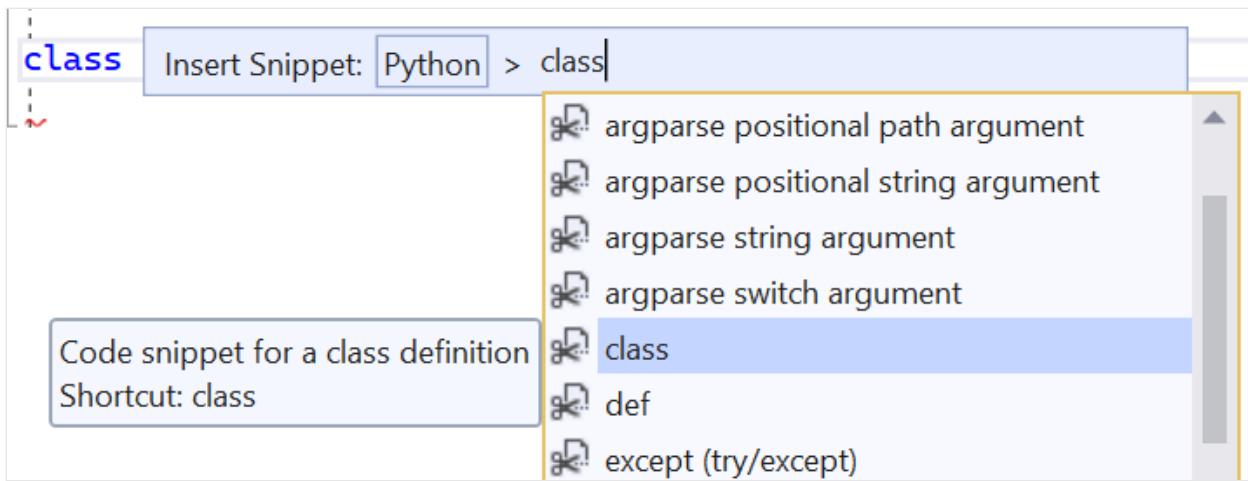
```
Module name
from xml.etree.cElementTree import ElementTree, ParseError
Class name
def parse_xml_from_file(filename):
    Function name
    try:
        Parameter name
        data = ElementTree().parse(filename)
    except ParseError:
        print('Failed to parse file {}'.format(filename))

filename = r'C:\Data.xml'
parse_xml_from_file(filename)
```

Inserir snippets de código

Os snippets de código são fragmentos de código que podem ser inseridos nos arquivos usando um atalho do teclado e pressionando **Tab**. Você também pode usar os comandos **Editar>IntelliSense>Inserir Snippet** e **Cercar com**, selecionar **Python** e escolher o snippet desejado.

Por exemplo, `class` é um atalho para um snippet de código que insere uma definição de classe. Você vê o snippet aparecer na lista de conclusão automática ao digitar `class:`

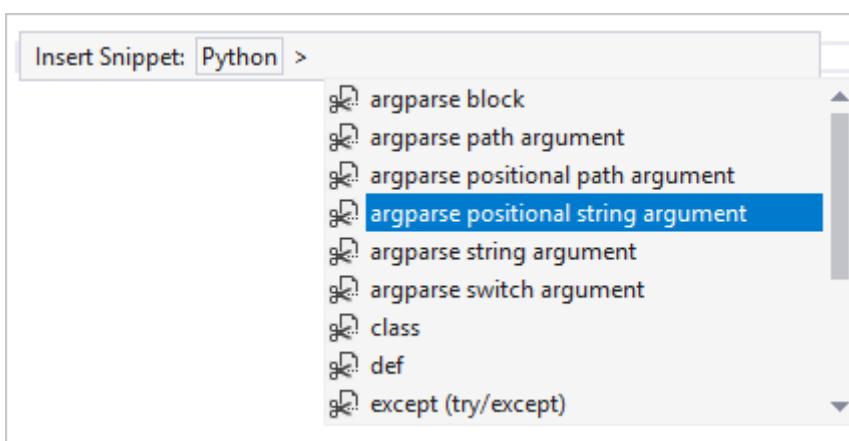


Pressionar **Tab** gera o restante da classe. Depois, digite o nome e a lista de bases, percorra os campos realçados com **Tab** e pressione **Enter** para começar a digitar o corpo.

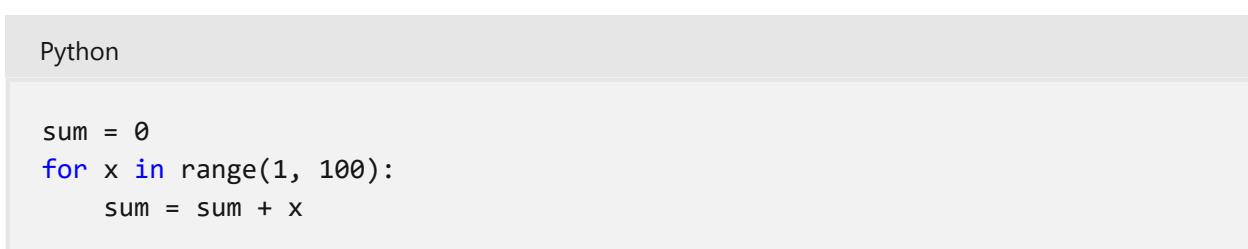
```
class ClassName(object):  
    pass
```

Comandos de menu

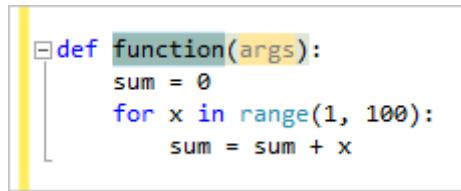
Ao usar o comando de menu **Editar>IntelliSense>Inserir Snippet de Código**, primeiro selecione **Python**, depois escolha o snippet desejado:



O comando **Editar>IntelliSense>Cercar com** coloca a seleção atual no editor de texto dentro de um elemento estrutural escolhido. Suponha que você tem um trecho de código semelhante ao seguinte exemplo:



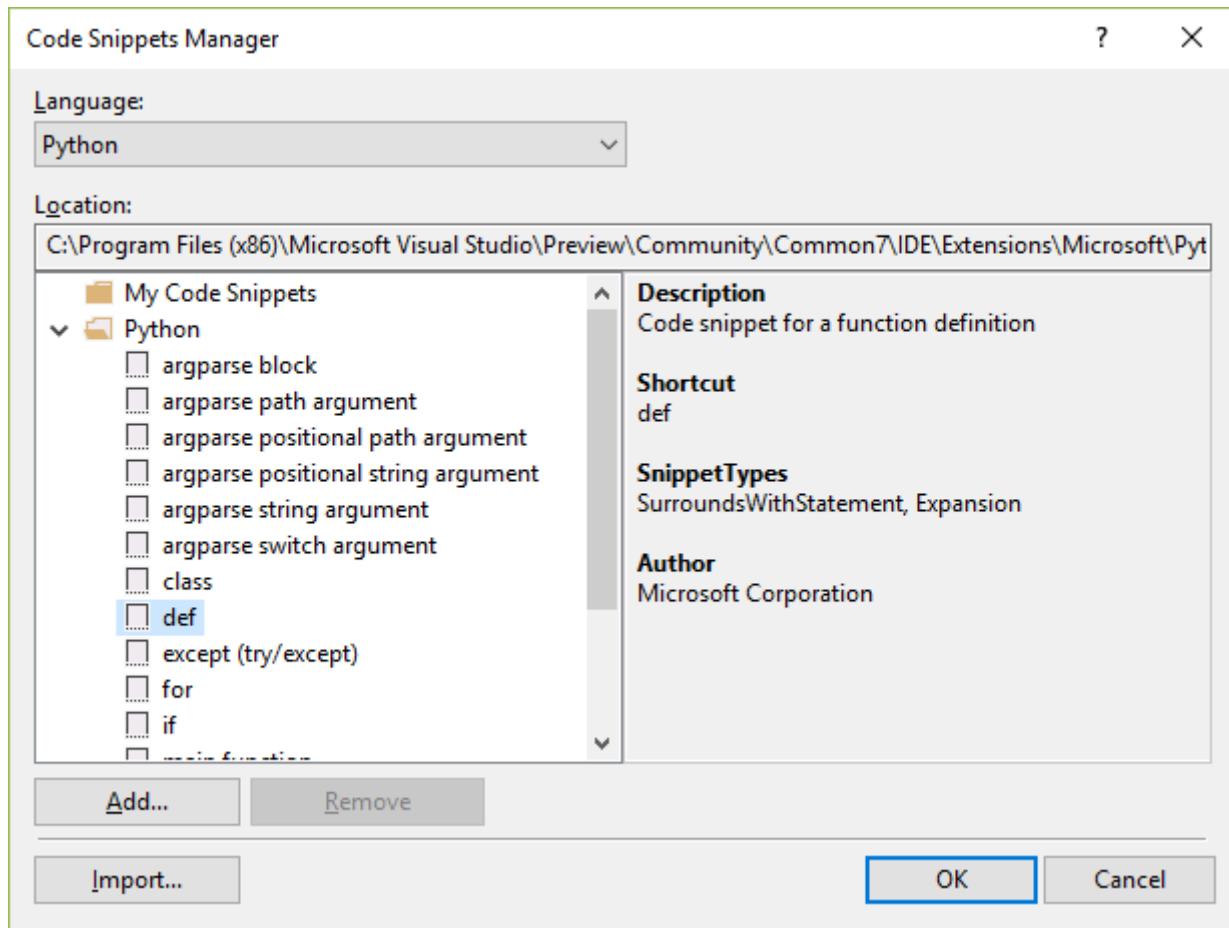
Selecionar esse código e escolher o comando **Envolver com** exibe uma lista de snippets de código disponíveis. A escolha de **def** na lista de snippets coloca o código selecionado em uma definição de função. Use a tecla **Tab** para navegar entre o nome e os argumentos realçados da função:



```
def function(args):
    sum = 0
    for x in range(1, 100):
        sum = sum + x
```

Examinar snippets disponíveis

Os snippets de código disponíveis podem ser visualizados no **Gerenciador de Snippets de Código**. Para acessar esse recurso, abra **Ferramentas>Gerenciador de Snippets de Código** e escolha **Python** como linguagem:



Para criar seus próprios snippets de código, confira [Passo a passo: Criar um snippet de código](#).

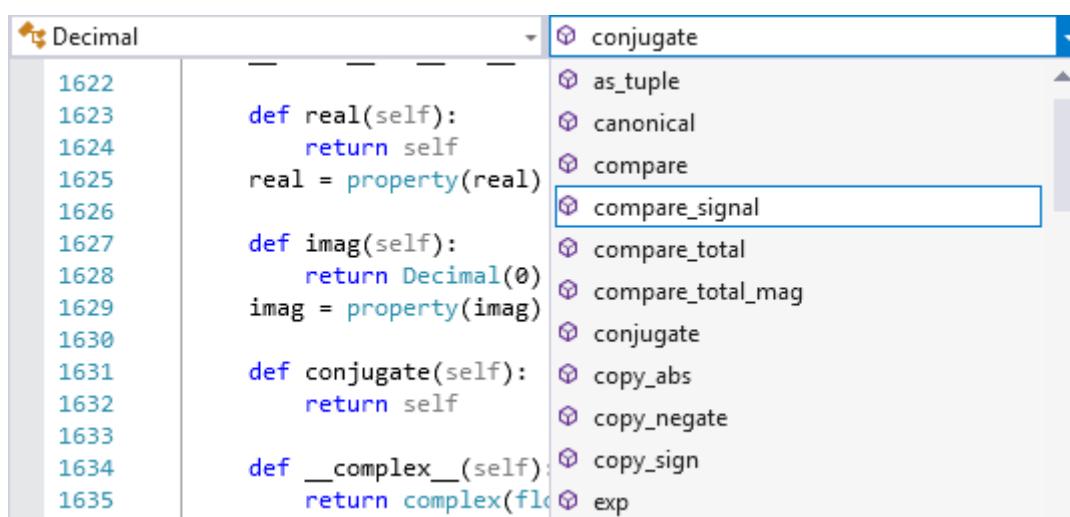
Se você escrever um ótimo snippet de código que gostaria de compartilhar, fique à vontade para postá-lo em linhas gerais e [contar para nós](#). Talvez possamos incluí-lo em uma versão futura do Visual Studio.

Navegar pelo seu código

O suporte ao Python no Visual Studio proporciona diversas formas de navegar rapidamente pelo código, incluindo bibliotecas com o código-fonte disponível. Há bibliotecas disponíveis com código-fonte para a [barra de navegação](#) e para os comandos [Ir para Definição](#), [Ir para](#) e [Localizar Todas as Referências](#). Use também o [Pesquisador de Objetos](#) do Visual Studio.

Barra de navegação

A barra de navegação é exibida na parte superior de cada janela do editor e inclui uma lista de dois níveis de definições. A lista suspensa do lado esquerdo apresenta definições de classes e funções de nível superior no arquivo atual. A lista suspensa do lado direito contém as definições dentro do escopo mostrado do lado esquerdo. Conforme você usa o editor, as listas são atualizadas para mostrar o contexto atual e você também pode selecionar uma entrada dessas listas para ir diretamente para ela.



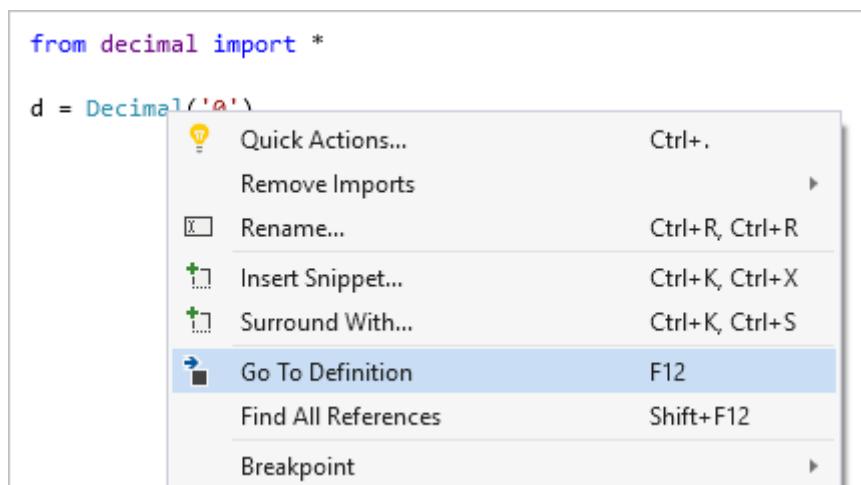
Dica

Para ocultar a barra de navegação, acesse **Ferramentas > Opções > Editor de Texto > Python > Gerais** e desmarque **Configurações > Barra de navegação**.

Ir para definição

O comando **Ir para Definição** vai rapidamente do uso de um identificador (como um nome de função, classe ou variável) para o local da definição do código-fonte. Para invocar o comando, clique com o botão direito do mouse em um identificador e escolha **Ir para Definição** ou posicione o cursor no identificador e pressione **F12**. O comando funciona em todo o código e nas bibliotecas externas em que o código-fonte esteja

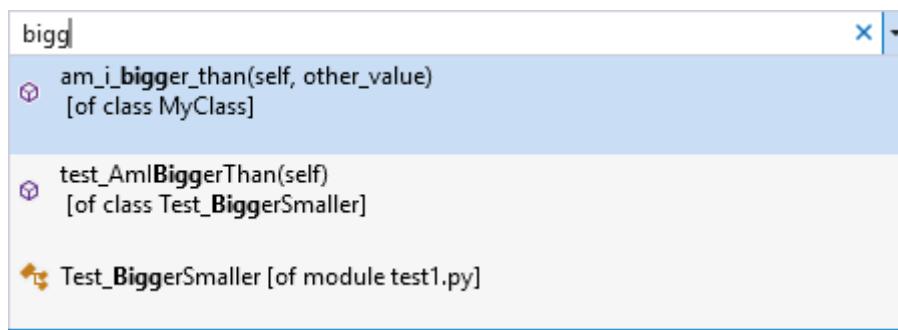
disponível. Se o código-fonte da biblioteca não estiver disponível, o comando **Ir para Definição** irá para a instrução `import` relevante de uma referência de módulo ou exibirá um erro.



Ir para

O comando **Editar>Ir para (Ctrl+,)** exibe uma caixa de pesquisa no editor em que é possível digitar qualquer cadeia de caracteres e ver as possíveis correspondências no código que definem uma função, uma classe ou uma variável que contém a cadeia de caracteres. Esse recurso fornece uma funcionalidade semelhante a **Ir Para Definição**, mas sem a necessidade de localizar um uso de um identificador.

Para navegar até a definição desse identificador, clique duas vezes em qualquer nome ou selecione o nome com as teclas de seta e pressione **Enter**.



Localizar Todas as Referências

O recurso **Localizar Todas as Referências** é uma maneira útil de descobrir o local em que um identificador específico é definido e usado, incluindo importações e atribuições. Para invocar o comando, clique com o botão direito do mouse em um identificador e escolha **Localizar Todas as Referências** ou posicione o cursor no identificador e pressione **Shift+F12**. Clicar duas vezes em um item da lista navegará para sua localização.

A screenshot of the PyCharm IDE interface. At the top, there is a code editor window containing Python code. A context menu is open over the line `d = Decimal('0')`, listing options like 'Quick Actions...', 'Remove Imports', 'Rename...', 'Insert Snippet...', 'Surround With...', 'Go To Definition', and 'Find All References'. The 'Find All References' option is highlighted with a blue selection bar. Below the code editor is a 'Find Symbol Results' tool window titled 'Find Symbol Results - 2 matches found'. It shows two entries under the 'References' tab: one pointing to the line in the current file and another pointing to the definition in the standard library module. There are also tabs for 'Values' and 'Definitions'.

```
from decimal import *
d = Decimal('0')
```

Conteúdo relacionado

- [Formatar o código do Python](#)
- [Refatorar um código do Python](#)
- [Usar um linter](#)

Comentários

Esta página foi útil?

[Yes](#)

[No](#)

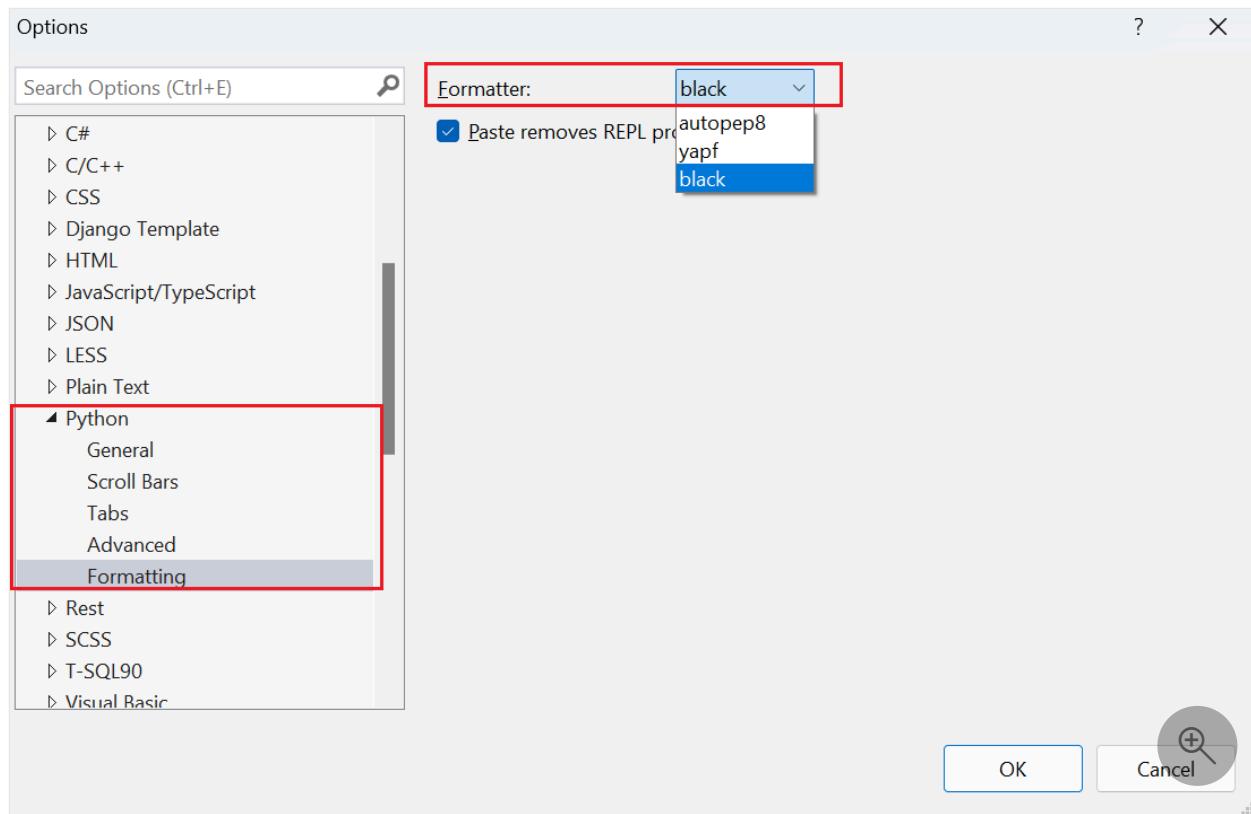
Reformatar automaticamente código Python no Visual Studio

Artigo • 18/04/2024

O Visual Studio permite que você reformate rapidamente o código para corresponder a padrões de formatadores específicos. Neste artigo, você vai explorar como acessar e habilitar recursos de formatação.

Escolher um formatador

Você pode definir o formatador de código-fonte por meio de **Ferramentas>Opções>Editor de texto>Python>Formatação**. As Ferramentas do Python no Visual Studio dão suporte à formatação de código-fonte com o autopep8, o formatador Black e o yapf.



O [suporte do Python no Visual Studio](#) também adiciona o comando **Preencher Parágrafo de Comentário** ao menu **Editar>Avançado**, conforme descrito em uma seção posterior.

Aplicar formato à seleção ou ao arquivo

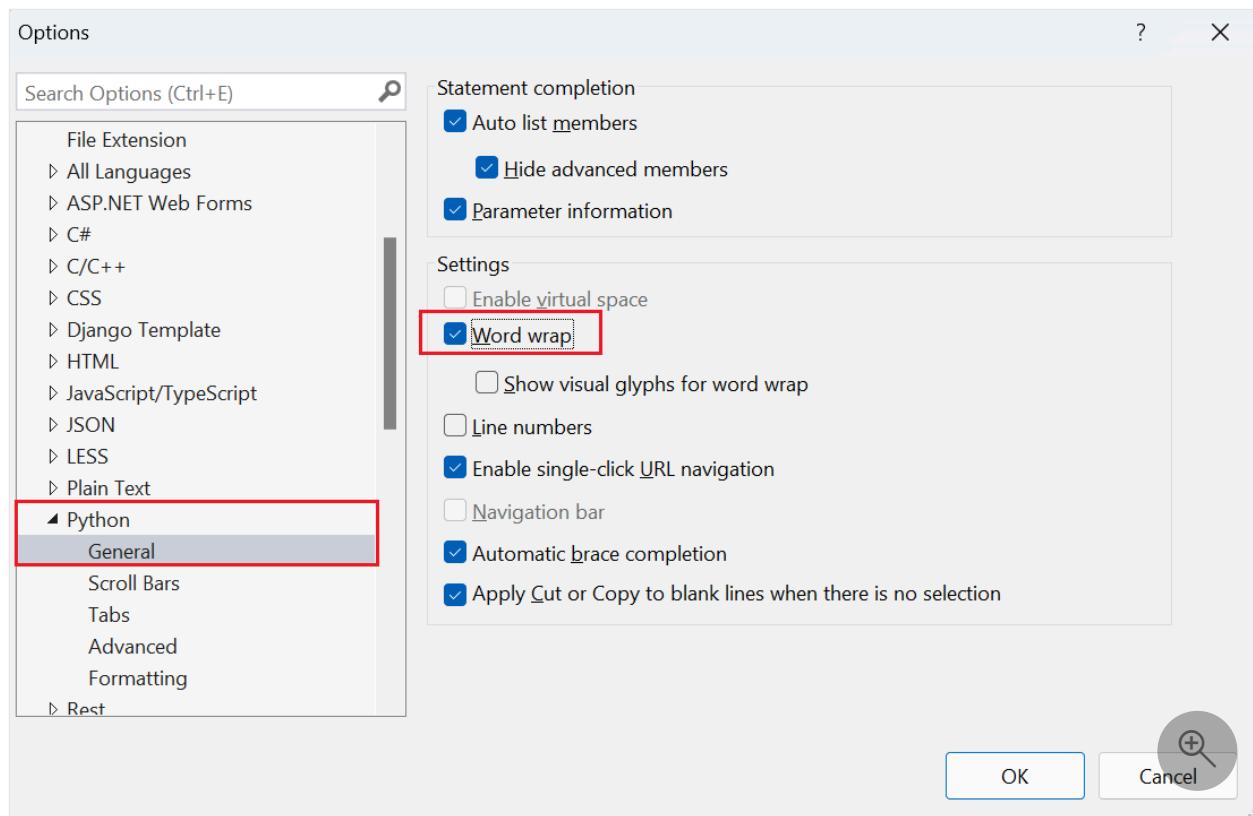
Você pode aplicar configurações de formatação a todo o conteúdo de um arquivo ou apenas a uma seleção específica.

Para formatar uma seleção, selecione **Editar>Avançado>Formatar Seleção**.

Para formatar todo o arquivo, selecione **Editar>Avançado>Formatar Documento**.

Quebra automática de linha

Você pode habilitar a quebra automática de linha em **Ferramentas>Opções>Editor de Texto>Python>Geral**. Na seção **Configurações**, marque a caixa de seleção **Quebra automática de linha**.



Formatar texto de comentário

A opção **Editar>Avançado>Preencher Parágrafo de Comentário** altera o fluxo e formata o texto do comentário.

Reformatar linhas longas

Use o recurso para dividir linhas longas de texto, conforme mostrado neste exemplo:

Python

```
# This is a very long  
long long long long long long long comment
```

O texto é reformatado como várias linhas:

Python

```
# This is a very long  
# long long long long long long comment
```

Combinar linhas curtas

Use o recurso para combinar linhas curtas de texto, conforme mostrado neste exemplo:

Python

```
# Short line of text  
# more text  
# text
```

O texto é reformatado em uma só linha:

Python

```
# Short line of text more text text
```

Conteúdo relacionado

- [Editar o código Python](#)
- [Refatorar um código do Python](#)
- [Executar lint no código Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Refatorar código Python no Visual Studio

Artigo • 18/04/2024

A reutilização de código existente e a atualização de código são tarefas comuns para desenvolvedores. Talvez você queira refatorar um código existente para dar outra finalidade a ele e economizar tempo por não precisar escrever um código novo do zero. Talvez você queira limpar um código com o intuito de retirar itens não usados ou atualizar importações e métodos para que sejam atualizados com as versões recentes.

O Visual Studio fornece vários comandos para ajudar a transformar e limpar código-fonte Python automaticamente:

- [Renomear](#) altera o nome de uma classe, um método ou uma variável.
- [Adicionar importação](#) fornece uma marcação inteligente para adicionar uma importação ausente.
- [Remover importações não utilizadas](#) exclui importações não utilizadas.

Pré-requisitos

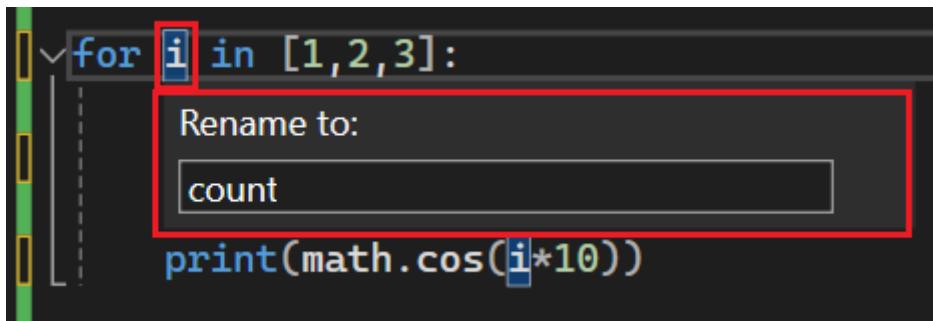
- Visual Studio. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).
- Acesso a um projeto com código Python existente.

Renomear uma classe, um método ou uma variável

Use o comando Renomear para modificar o nome de um identificador, incluindo uma classe, um método ou uma variável. O Visual Studio é compatível tanto com a atualização de todas as instâncias do identificador quanto com a atualização de instâncias específicas que você indicar.

As etapas a seguir mostram como utilizar o comando **Renomear** em um código.

1. No código, clique com o botão direito do mouse no identificador que deseja renomear e escolha **Renomear**. Também é possível posicionar o cursor em um identificador e escolher **Editar>Refatorar>Renomear** no menu ou utilizar o atalho do teclado **Ctrl+R..**
2. Na caixa de diálogo **Renomear**, digite o novo nome do identificador e pressione **Enter**:

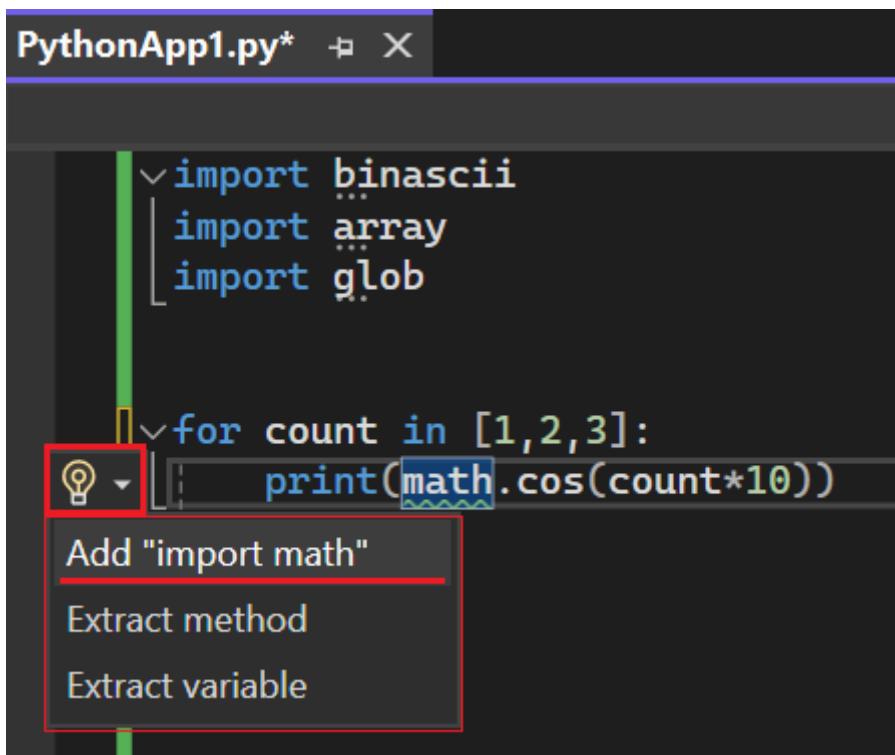


Adicionar uma instrução de importação

Caso existam identificadores sem definições ou informações complementares sobre tipo no código, o Visual Studio poderá ajudar você a corrigir o problema. Posicione o cursor em um identificador sem informações para que o Visual Studio mostre uma marcação inteligente (lâmpada) à esquerda do código. A marcação lista comandos para adicionar as instruções `import` ou `from ... import` necessárias para o identificador correspondente.

As etapas a seguir mostram como usar a marcação inteligente a fim de adicionar importações ao código.

1. No código, posicione o cursor em um identificador para o qual o Visual Studio exibe a marcação inteligente (lâmpada). Nesse exemplo, a marcação inteligente é exibida para a chamada do módulo `math`:



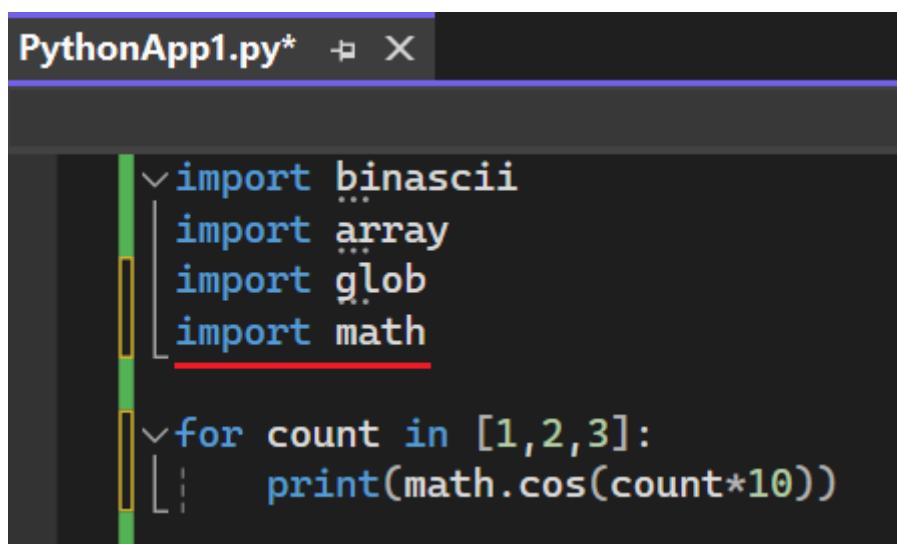
2. No menu da marcação inteligente, escolha o comando para adicionar o módulo necessário ou digite informações no arquivo do código. Nesse exemplo, o

comando para adicionar a instrução `import math` é selecionado.

O Visual Studio oferece conclusões `import` para módulos e pacotes de nível superior no projeto atual e na biblioteca padrão. O Visual Studio também oferece conclusões `from ... import` para submódulos e subpacotes, bem como para membros do módulo. As conclusões incluem funções, classes e dados exportados.

3. Após selecionar uma opção, confirme se a alteração esperada acontece no arquivo.

O Visual Studio adiciona a instrução `import` no topo do arquivo de código após outras importações ou em uma instrução `from ... import` existente se o mesmo módulo já foi importado. Nesse exemplo, a instrução `import math` é adicionada no topo do arquivo, depois das outras importações:



A screenshot of a Python code editor window titled "PythonApp1.py*". The code shown is:

```
PythonApp1.py*  ✎ X

import binascii
import array
import glob
import math

for count in [1,2,3]:
    print(math.cos(count*10))
```

The word "math" under the first import statement is highlighted in red, indicating it is a suggestion or a member being completed. The code editor interface shows standard syntax highlighting for Python, with blue for keywords and green for comments.

O Visual Studio tenta filtrar os membros que não estão definidos em nenhum módulo. Um exemplo é um módulo importado em outro módulo que não é filho do módulo importador. Vários módulos utilizam a instrução `import sys` em vez de `from xyz import sys`. Você não vê a conclusão da importação do módulo `sys` de outros módulos, mesmo que os módulos não tenham um membro `_all_` que exclua o módulo `sys`.

Da mesma forma, o Visual Studio filtra funções importadas de outros módulos ou do namespace interno. Se um módulo importa a função `settrace` do módulo `sys`, em teoria, é possível importá-la desse módulo. Entretanto, a melhor abordagem é utilizar a instrução `import settrace from sys` diretamente, por isso o Visual Studio oferece especificamente essa instrução.

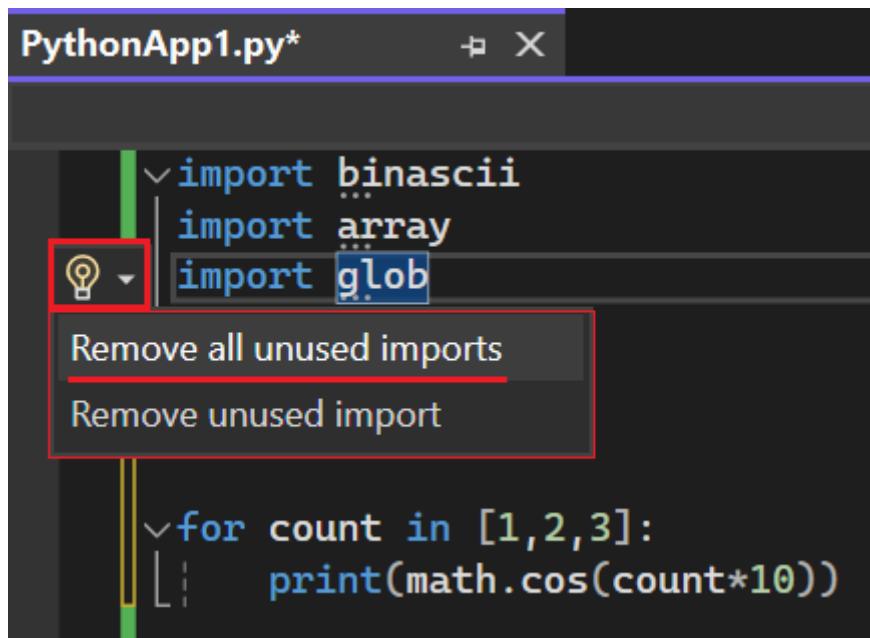
Por fim, suponha que um módulo normalmente é excluído, mas ele tem outros valores que são incluídos, como um nome atribuído com um valor no módulo. O Visual Studio exclui a importação mesmo assim. Esse comportamento pressupõe que o valor não deve ser exportado porque outro módulo o define. É provável que outra atribuição seja um valor fictício que também não é exportado.

Remover importações não utilizadas

Ao escrever código, é fácil acabar tendo instruções `import` para módulos que não estão sendo usadas. Como o Visual Studio analisa o código, ele pode determinar automaticamente se uma instrução `import` é necessária, observando se o nome importado é usado dentro do escopo abaixo, no qual a instrução ocorre.

As etapas a seguir mostram como eliminar importações não usadas no código.

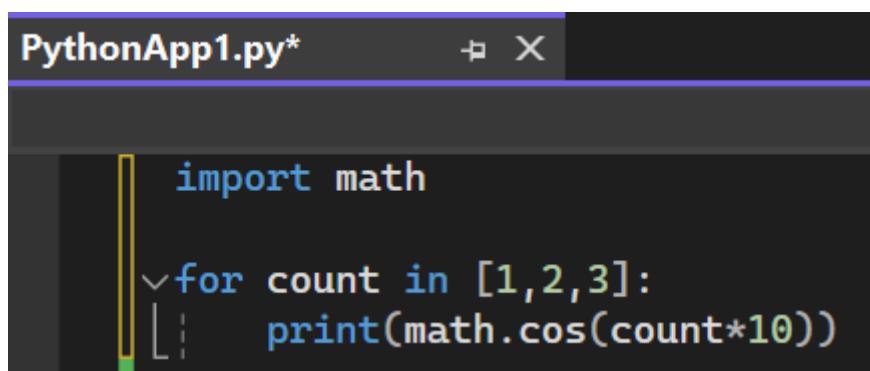
1. No código, posicione o cursor em uma instrução `import` para a qual o Visual Studio exibe a marcação inteligente (lâmpada). Nesse exemplo, a marcação inteligente é exibida para os módulos não usados `binascii`, `array` e `glob`:



The screenshot shows a Python code editor window titled "PythonApp1.py*". In the code, there are three import statements: "import binascii", "import array", and "import glob". The "glob" import statement has a lightbulb icon with a dropdown arrow next to it, indicating it is unused. A context menu is open at this location, with the "Remove all unused imports" option highlighted with a red border. The menu also contains "Remove unused import". Below the imports, there is a for loop that prints the cosine of each number from 1 to 3.

```
PythonApp1.py*
import binascii
import array
import glob
Remove all unused imports
Remove unused import
for count in [1,2,3]:
    print(math.cos(count*10))
```

2. Escolha as opções **Excluir todas as importações não usadas** ou **Remover importação não usada** para eliminar apenas o módulo selecionado.
3. Após selecionar uma opção, confirme se as alterações acontecem no arquivo. Nesse exemplo, o Visual Studio elimina os três módulos não usados: `binascii`, `array` e `glob`.



The screenshot shows the same Python code editor window after the unused imports were removed. Only the "import math" statement remains, while the other three import statements have been removed.

```
PythonApp1.py*
import math
for count in [1,2,3]:
    print(math.cos(count*10))
```

Considerações a respeito do uso dos comandos de refatoração

Antes de usar os comandos de refatoração, examine as seguintes considerações.

- Após a execução de um comando de refatoração, é possível reverter as alterações usando o comando **Editar>Desfazer**. O comando **Renomear** oferece um recurso **Pré-visualizar**, para que você possa ver as alterações antes que elas sejam aplicadas.
- O Visual Studio não considera o fluxo de controle no código. Se você utilizar um identificador antes que a definição complementar exista no código, como uma instrução `import`, o Visual Studio processará o identificador como utilizado mesmo assim. O Visual Studio espera encontrar definições complementares para os identificadores antes que você faça chamadas e atribuições.
- O Visual Studio ignora todas as instruções de importação `from __future__`. Essas instruções são importações executadas dentro de uma definição de classe ou por meio de instruções `from ... import *`.

Conteúdo relacionado

- [Editar o código Python](#)
- [Lint de código Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Executar lint no código Python no Visual Studio

Artigo • 18/04/2024

O processo de lint destaca problemas de sintaxe e estilo da linguagem no código-fonte Python. É possível executar um linter no código a fim de identificar e corrigir erros sutis de programação ou práticas de codificação não convencionais que possam gerar erros. O lint é capaz de detectar o uso de uma variável não inicializada ou indefinida, chamadas para funções indefinidas, ausência de parênteses e outros problemas sutis, como tentativas de redefinição de tipos ou funções internos. O lint é diferente da formatação porque analisa a forma como o código é executado e detecta erros, enquanto a formatação apenas reestrutura a forma como o código é exibido.

Duas opções comuns de lint incluem [PyLint](#) e [MyPy](#). Essas ferramentas são amplamente usadas para verificar erros em código Python e incentivar bons padrões de codificação em Python. Ambas são integradas em projetos do Visual Studio para Python.

Pré-requisitos

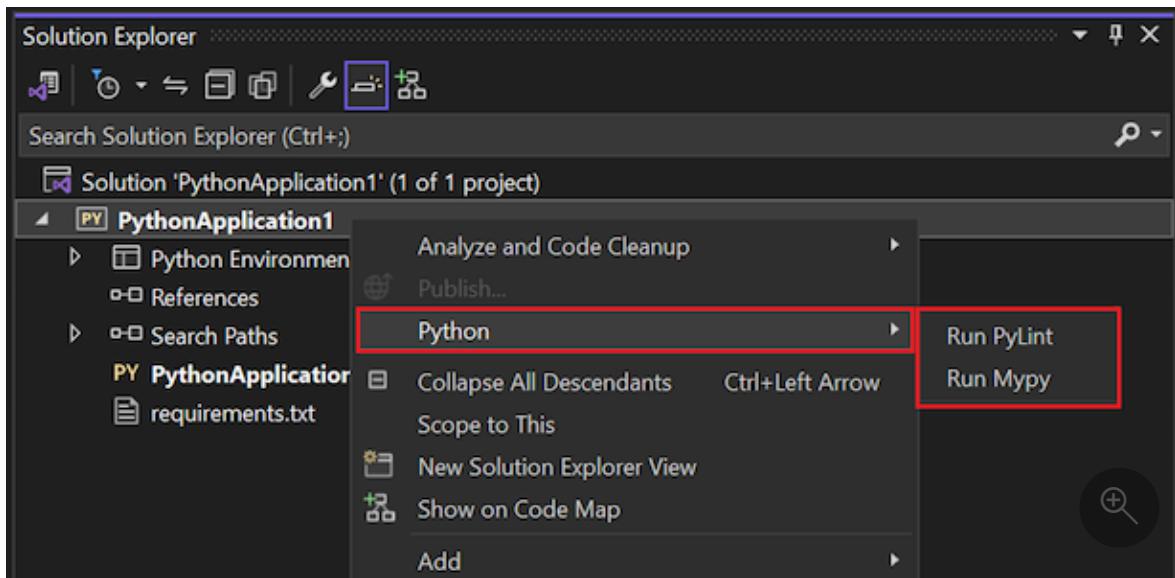
- Visual Studio. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).
- Acesso a um projeto do Python para execução de ferramentas de lint em código existente.

Executar uma ferramenta de lint

As ferramentas de lint do Visual Studio estão disponíveis no **Gerenciador de Soluções**.

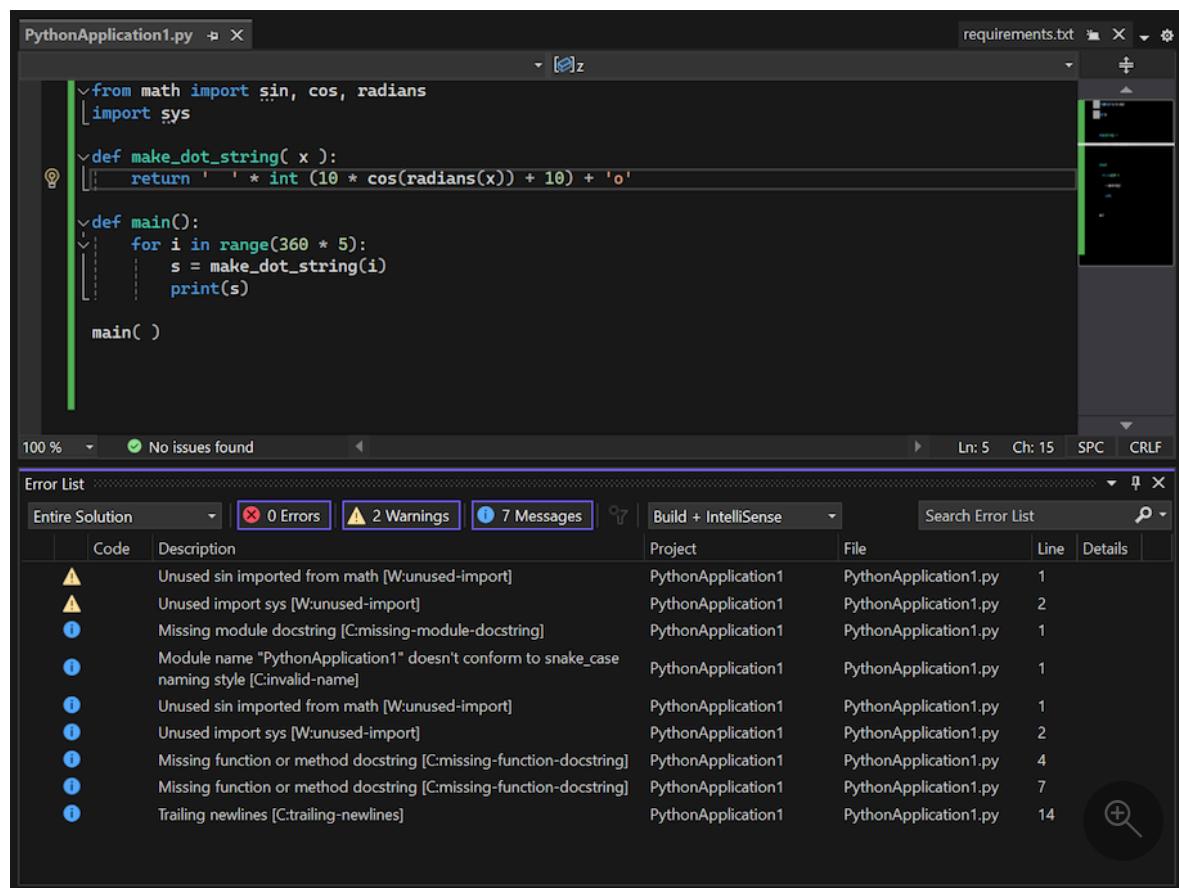
As etapas a seguir mostram como utilizar um linter para verificar o código.

1. No Visual Studio, clique com o botão direito do mouse em um projeto do Python no **Gerenciador de Soluções** e escolha **Python**, depois selecione **Executar PyLint** ou **Executar MyPy**:



O comando solicita que você instale o linter escolhido no ambiente ativo, caso ainda não esteja presente.

2. Após a execução do linter no código, analise todos os avisos e erros de lint na janela Lista de Erros:



3. Clique duas vezes em um erro ou aviso para acessar o local do problema no código-fonte.

Configurar opções de linha de comando

O [PyLint](#) e o [MyPy](#) oferecem opções de linha de comando para definir as configurações de lint para seu projeto.

Esta seção mostra um exemplo que utiliza as [opções de linha de comando](#) do PyLint para controlar o comportamento do PyLint por meio de um arquivo de configuração `.pylintrc`. Esse arquivo pode ser colocado na raiz de um projeto do Python no Visual Studio ou em outra pasta, dependendo da abrangência desejada para aplicação das configurações.

As etapas a seguir suprimem os avisos "docstring ausente" (conforme mostrado na imagem anterior) usando um arquivo `.pylintrc` no projeto do Python.

1. Na linha de comando, navegue até a pasta raiz do projeto que contém o arquivo `.pyproj` e execute o seguinte comando para gerar um arquivo de configuração comentado:

```
Console  
pylint --generate-rcfile > .pylintrc
```

2. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto e selecione **Adicionar>Item Existente**.
3. Na caixa de diálogo, acesse a pasta que contém o novo arquivo `.pylintrc`.
Selecione o arquivo `.pylintrc` e **Adicionar**.
4. No **Gerenciador de Soluções**, abra o arquivo `.pylintrc` para edição.
5. Você pode definir diversas configurações no arquivo. Esse exemplo mostra como desativar um aviso.
 - a. Encontre a seção `[MESSAGES CONTROL]`, depois encontre a configuração `disable` dentro dessa seção.

```
.pylintrc ▾ X
403
404 [MESSAGES CONTROL]
405
406 # Only show warnings with the listed confidence levels. Leave empty to show
407 # all. Valid levels: HIGH, CONTROL_FLOW, INFERENCE, INFERENCE_FAILURE,
408 # UNDEFINED.
409 confidence=HIGH,
410         CONTROL_FLOW,
411         INFERENCE,
412         INFERENCE_FAILURE,
413         UNDEFINED
414
415 # Disable the message, report, category or checker with the given id(s). You
416 # can either give multiple identifiers separated by comma (,) or put this
417 # option multiple times (only on the command line, not in the configuration
418 # file where it should appear only once). You can also use "--disable=all" to
419 # disable everything first and then re-enable specific checks. For example, if
420 # you want to run only the similarities checker, you can use "--disable=all"
421 # --enable=similarities". If you want to run only the classes checker, but have
422 # no Warning level messages displayed, use "--disable=all --enable=classes"
423 # --disable=W".
424 disable=raw-checker-failed,
425         bad-inline-option,
426         locally-disabled,
427         file-ignored,
428         suppressed-message,
429         useless-suppression,
430         deprecated-pragma,
431         use-symbolic-message-instead,
432         use-implicit-booleanness-not-comparison-to-string,
433         use-implicit-booleanness-not-comparison-to-zero
434
435 # Enable the message, report, category or checker with the given id(s). You can
436 # either give multiple identifier separated by comma (,) or put this option
```

A configuração `disable` consiste em uma longa cadeia de mensagens específicas, à qual é possível acrescentar os avisos desejados.

- b. Anexe a cadeia `,missing-docstring` (incluindo a vírgula) ao valor da configuração `disable`:

```
424     disable=raw-checker-failed,
425         bad-inline-option,
426         locally-disabled,
427         file-ignored,
428         suppressed-message,
429         useless-suppression,
430         deprecated-pragma,
431         use-symbolic-message-instead,
432         use-implicit-booleanness-not-comparison-to-string,
433         use-implicit-booleanness-not-comparison-to-zero,
434         missing-docstring
```

6. Salve o arquivo `.pylintrc`.

7. Execute o PyLint novamente. Observe que os avisos de “docstring ausente” foram suprimidos.

Utilizar o arquivo `pylintrc` do compartilhamento de rede

É possível utilizar um arquivo `.pylintrc` de um compartilhamento de rede.

1. Crie uma variável de ambiente chamada `PYLINTRC`.
2. Atribua à variável o valor do nome de arquivo no compartilhamento de rede utilizando um caminho UNC ou uma letra de unidade mapeada. Por exemplo,
`PYLINTRC=\myshare\python\.pylintrc`.

Conteúdo relacionado

- [Editar o código Python](#)
- [Refatorar um código do Python](#)

Comentários

Esta página foi útil?

 Yes

 No

Definir comandos personalizados para projetos em Python no Visual Studio

Artigo • 23/04/2024

Ao desenvolver os projetos em Python, é possível realizar uma comutação para uma janela de comando com a finalidade de executar scripts ou módulos específicos, realizar a execução de comandos pip ou usar outras ferramentas com seu código. Para aprimorar o fluxo de trabalho, é possível adicionar comandos personalizados ao menu do projeto **Python** no Visual Studio. Os comandos em Python personalizados podem ser executados em uma janela do console ou na janela de **Saída** do Visual Studio. Além disso, é possível usar expressões regulares para fornecer instruções ao Visual Studio sobre como analisar erros e avisos da saída do comando.

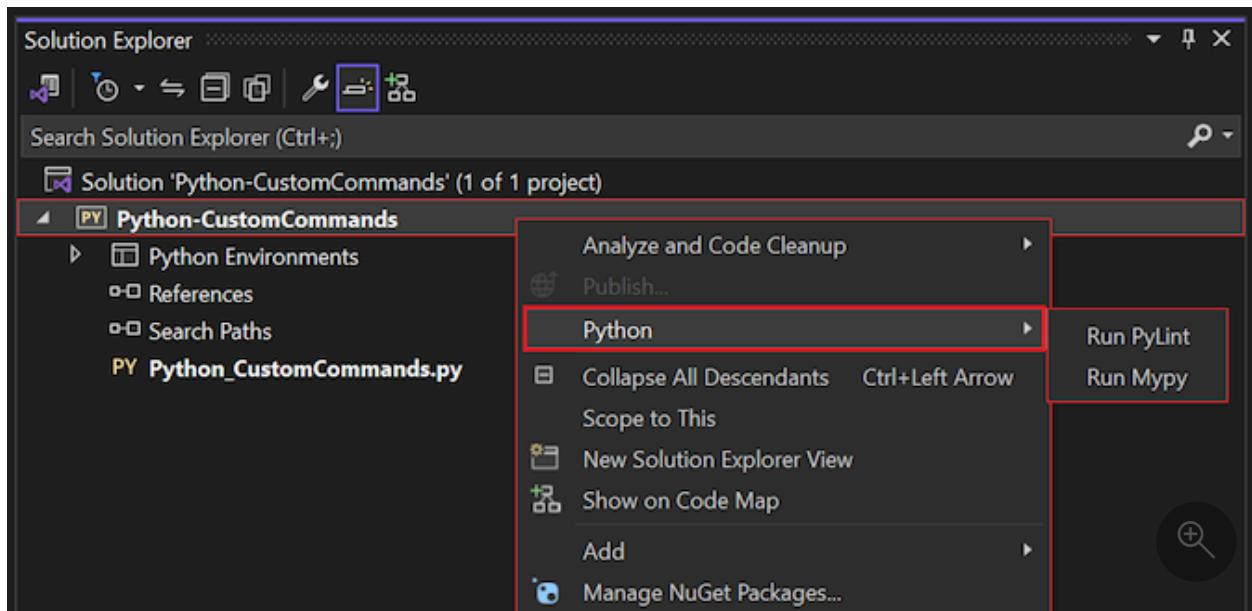
Pré-requisitos

- Instalação do Visual Studio no Windows com suporte para cargas de trabalho em Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis ↗](#).

Explorar comandos personalizados

Por padrão, o menu do projeto em Python contém dois comandos, **Executar PyLint** e **Executar Mypy**:

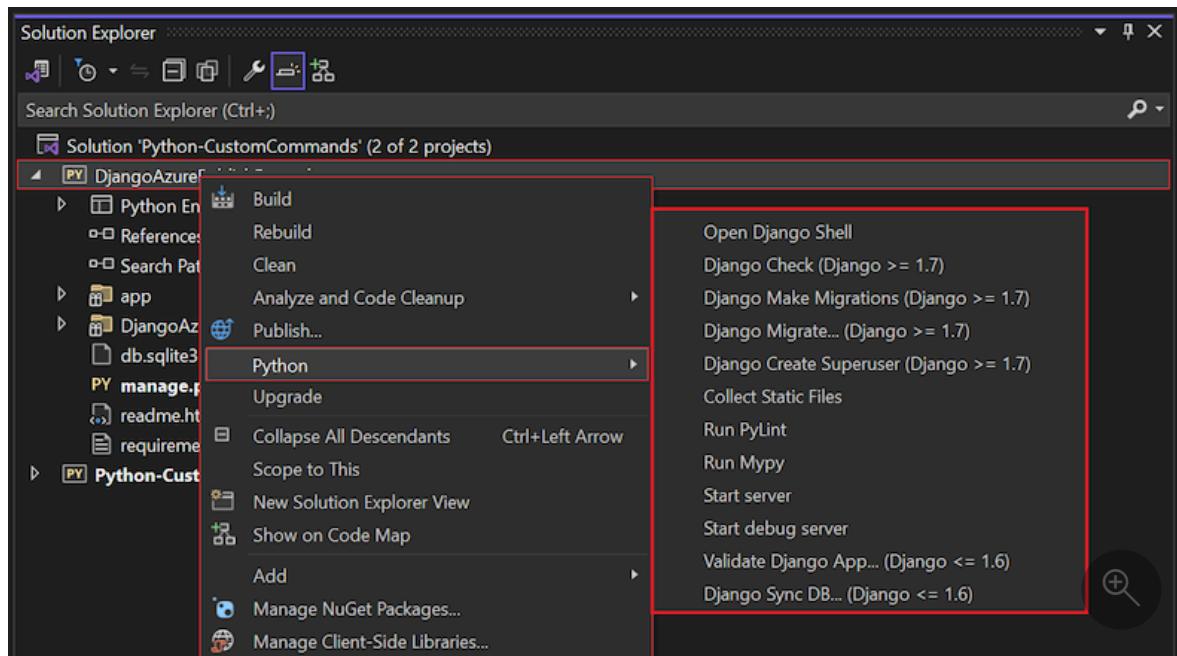


Os comandos em Python personalizados que você definir aparecerão nesse mesmo menu. Um comando personalizado pode fazer referência a um arquivo em Python, um módulo de Python, um código Python embutido, um executável arbitrário ou um comando pip. Você também pode especificar como e onde o comando é executado.

É possível adicionar comandos personalizados de diversas maneiras:

- Definir comandos personalizados diretamente em um arquivo de projeto em Python (*.pyproj*). Esses comandos se aplicam a esse projeto específico.
- Definir comandos personalizados em um arquivo de destino (*.targets*). É possível importar os comandos com facilidade nesse arquivo para usá-los em vários projetos.
- Criar um projeto em Python usando um modelo de projeto no Visual Studio que defina comandos em Python personalizados.

Determinados modelos de projeto em Python no Visual Studio adicionam comandos personalizados ao usar um arquivo de destino. Os modelos Projeto Web em Bottle e Projeto Web em Flask adicionam dois comandos, **Iniciar servidor** e **Iniciar servidor de depuração**. O modelo Projeto Web em Django adiciona estes comandos e vários outros:



Recarregar o projeto para obter acesso aos comandos personalizados

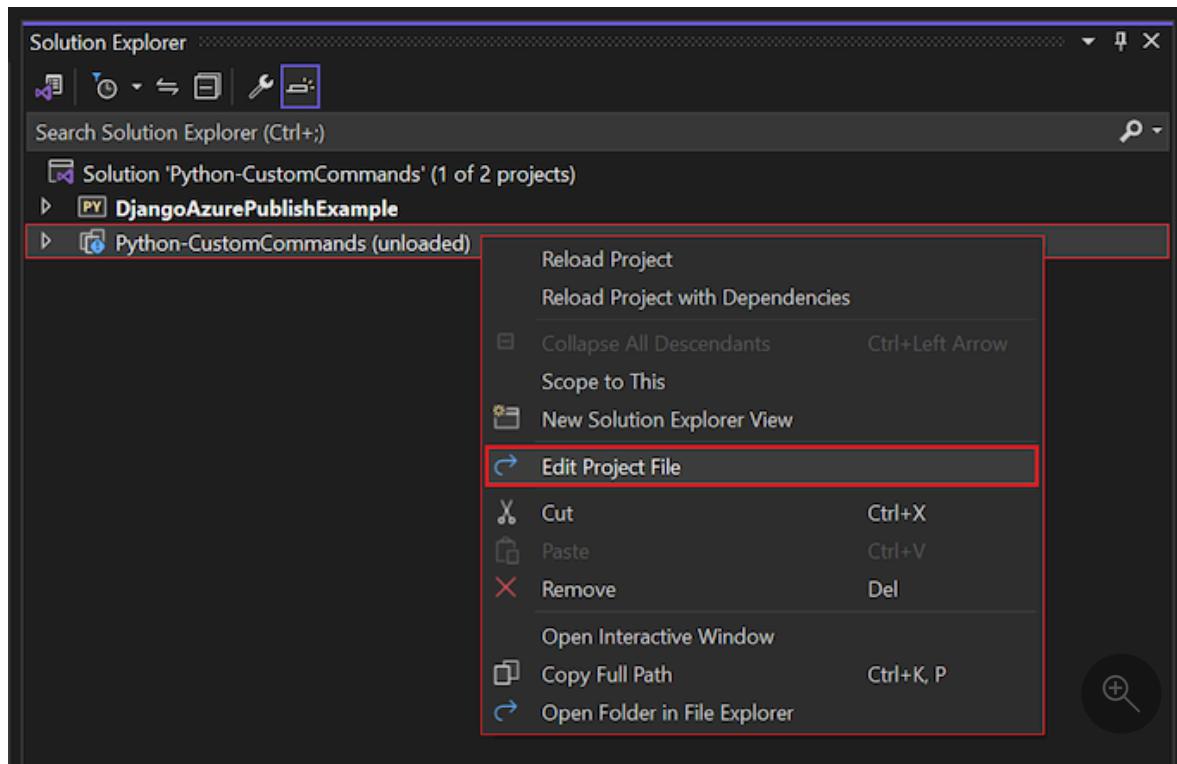
Quando um projeto é aberto no Visual Studio, se você fizer alterações no arquivo de projeto correspondente em um editor, deverá recarregar o projeto para aplicar as alterações. De maneira semelhante, após definir comandos de Python personalizados em um arquivo de projeto em Python, você precisa recarregar o projeto em Python para que os comandos apareçam no menu do projeto em **Python**. Ao modificar comandos personalizados definidos em um arquivo de destino, é necessário recompilar a solução completa do Visual Studio para projetos que importem esse arquivo de destino.

Uma abordagem comum é fazer alterações no arquivo de projeto em Python diretamente no Visual Studio:

1. Abra o projeto em Python no Visual Studio. (Ao abrir um projeto no Visual Studio, o projeto é *carregado* por padrão.)
2. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto em Python e selecione **Descarregar Projeto**.

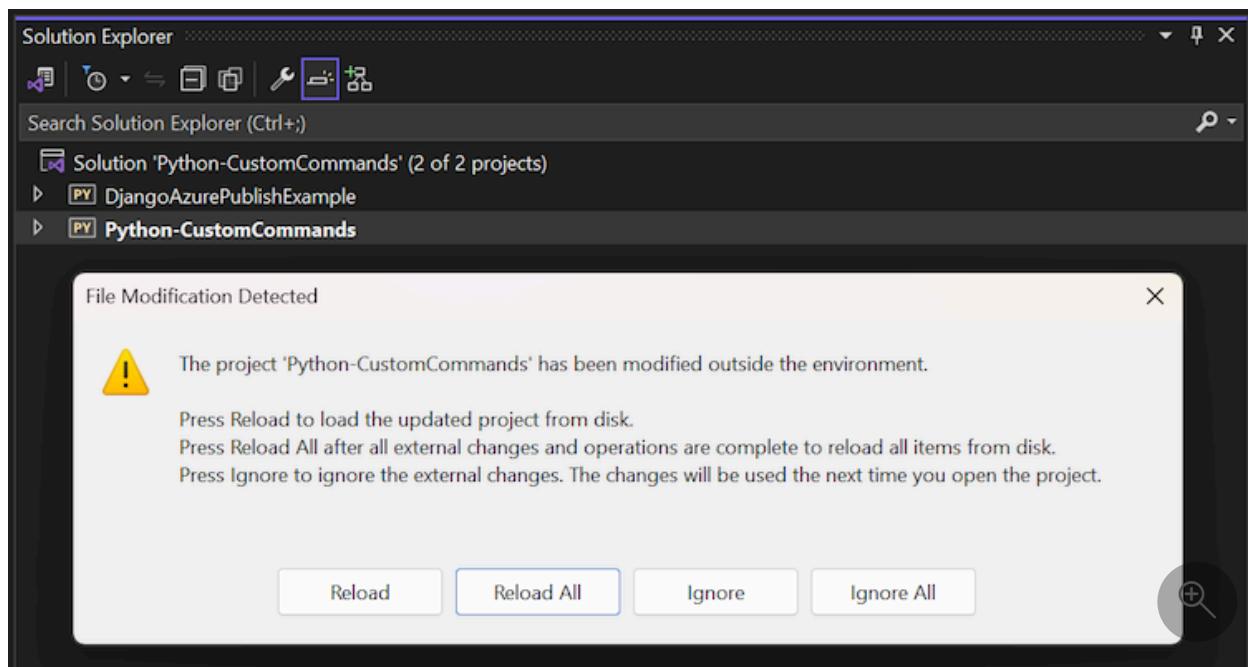
O Visual Studio descarrega o projeto e abre o arquivo de projeto em Python correspondente (*.pyproj*) no editor.

Se o arquivo de projeto não abrir, clique com o botão direito do mouse no projeto em Python novamente e selecione **Editar Arquivo de Projeto**:



3. Faça as alterações no arquivo de projeto no editor do Visual Studio e salve o trabalho.
4. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto descarregado e selecione **Recarregar Projeto**. Se você tentar recarregar o projeto sem salvar as alterações no arquivo de projeto, o Visual Studio solicitará que você conclua a ação.

O processo de descarregar, editar, salvar e recarregar pode se tornar tedioso quando você desenvolve comandos personalizados. Um fluxo de trabalho mais eficiente envolve carregar simultaneamente o projeto no Visual Studio e abrir o arquivo de projeto em Python em um editor separado. É possível usar qualquer editor, como outra instância do Visual Studio, o Visual Studio Code, o Bloco de notas do Windows, e assim por diante. Após salvar as alterações no editor e retornar para o Visual Studio, o Visual Studio detecta as alterações no arquivo de projeto para o projeto aberto e solicita que você execute uma ação:



Selecione Recarregar ou Recarregar Tudo e o Visual Studio aplicará imediatamente as alterações do arquivo de projeto ao projeto aberto.

Adicionar comandos personalizados com um arquivo de projeto

O procedimento apresentado a seguir mostra como criar um comando personalizado ao adicionar a definição no arquivo de projeto em Python (`.pyproj`) e recarregar o projeto no Visual Studio. O comando personalizado realiza a execução do arquivo de inicialização de um projeto, de forma direta, ao usar o comando `python.exe`, que é basicamente o mesmo que usar a opção **Depurar>Iniciar sem a Depuração** na barra de ferramentas principal do Visual Studio.

1. No Visual Studio, crie um novo projeto em Python chamado *Python-CustomCommands* ao usar o modelo **Aplicativo do Python**. Para obter instruções, confira [Quickstart: Create a Python project from a template](#).

O Visual Studio cria o projeto em Python e o carrega em sua sessão. É possível configurar o projeto por meio do arquivo de projeto (`.pyproj`). Este arquivo fica visível somente no Visual Studio quando o projeto está aberto, mas *descarregado*. Além disso, o projeto tem um arquivo em Python (`.py`) para o código do aplicativo.

2. Abra o arquivo de aplicativo *Python_CustomCommands.py* no editor e adicione o seguinte código:

```
Python
```

```
print("Hello custom commands")
```

3. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto em Python, selecione **Python** e observe os comandos no menu de contexto. Atualmente, os únicos comandos no menu de contexto são **Executar PyLint** e **Executar Mypy**. Ao definir comandos personalizados, eles também aparecerão nesse menu.
4. Inicie um editor separado externo à sessão do Visual Studio e abra o arquivo de projeto em Python (*Python-CustomCommands.pyproj*) no editor. (Certifique-se de abrir o arquivo de projeto [.pyproj] e não o arquivo de aplicativo em Python [.py].)
5. No arquivo de projeto, localize o elemento de fechamento `</Project>` no final do arquivo e adicione o seguinte XML imediatamente antes do elemento de fechamento:

XML

```
<PropertyGroup>
  <PythonCommands>
    $(PythonCommands);
  </PythonCommands>
</PropertyGroup>
```

6. Salve as alterações no arquivo de projeto e retorne para o Visual Studio. O Visual Studio detecta as alterações no arquivo de projeto e solicita que você execute uma ação. Na solicitação, selecione **Recarregar** para atualizar o projeto aberto com as alterações do arquivo de projeto.
7. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto em Python, selecione **Python** e verifique os comandos no menu de contexto.

O menu de contexto ainda mostra somente os comandos **Executar PyLint** e **Executar Mypy**. O código que você acabou de adicionar ao arquivo de projeto simplesmente replica o grupo de propriedade padrão `<PythonCommands>` que contém o comando **PyLint**. Na próxima etapa, você adiciona mais código para o comando personalizado.

8. Realize a comutação para o editor no qual você está realizando a atualização do arquivo de projeto. Adicione a definição de elemento `<Target>` apresentada a seguir no elemento `<Project>`. É possível posicionar a definição `<Target>` antes ou depois da definição `<PropertyGroup>` descrita anteriormente.

Esse elemento `<Target>` define um comando personalizado para a execução do arquivo de inicialização para o projeto (identificado pela propriedade `StartupFile`) ao usar o comando `python.exe` em uma janela do console. A definição do atributo `ExecuteIn="consolepause"` usa um console que aguarda a seleção de uma chave para fechar a janela do console.

XML

```
<Target Name="Example_RunStartupScript" Label="Run startup file"  
Returns="@.Commands">  
  <CreatePythonCommandItem  
    TargetType="script"  
    Target="$(StartupFile)"  
    Arguments=""  
    WorkingDirectory="$(MSBuildProjectDirectory)"  
    ExecuteIn="consolepause">  
    <Output TaskParameter="Command" ItemName="Commands" />  
  </CreatePythonCommandItem>  
</Target>
```

9. Substitua o grupo de propriedades `<PythonCommands>` (adicionado na etapa 5) pelo XML apresentado a seguir. Essa sintaxe define o atributo `Name` para o elemento `<Target>`, que adiciona o comando personalizado ao menu de contexto do Python. O comando tem o rótulo do menu **Executar arquivo de inicialização**.

XML

```
<PythonCommands>  
  $(PythonCommands);  
  Example_RunStartupScript  
</PythonCommands>
```

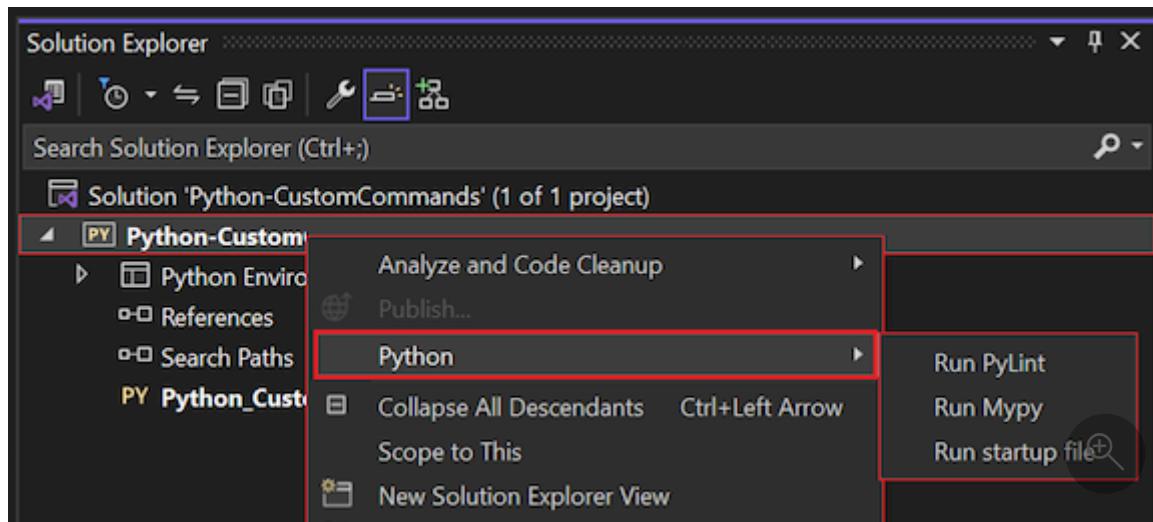
Dica

Se você deseja que o comando personalizado apareça no menu de contexto antes dos comandos padrão definidos no token `$(PythonCommands)`, posicione a sintaxe `<Target>` do comando antes desse token.

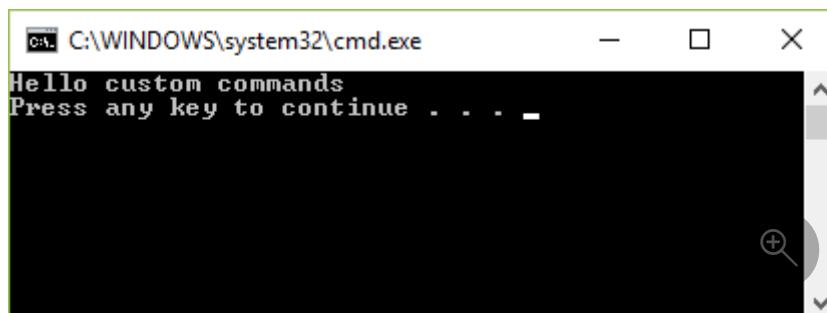
10. Salve as alterações no arquivo de projeto e retorne para o Visual Studio. Na solicitação, recarregue o projeto.

11. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto em Python, selecione **Python** e verifique novamente os comandos no menu de contexto.

Agora o comando personalizado **Executar arquivo de inicialização** está no menu. Se você não vir o comando personalizado, confirme se adicionou o valor de atributo `Name` do elemento `<Target>` ao elemento `<PythonCommands>`, conforme descrito na [etapa 9](#). Além disso, realize uma revisão das considerações listadas na seção [Solução de problemas](#) apresentada posteriormente neste artigo.



12. Selecione o comando **Executar arquivo de inicialização**. Uma janela do console é aberta e exibe o texto **Boas-vindas aos comandos personalizados** seguido de **Pressione qualquer tecla para continuar**. Confirme a saída e feche a janela do console.



Observação

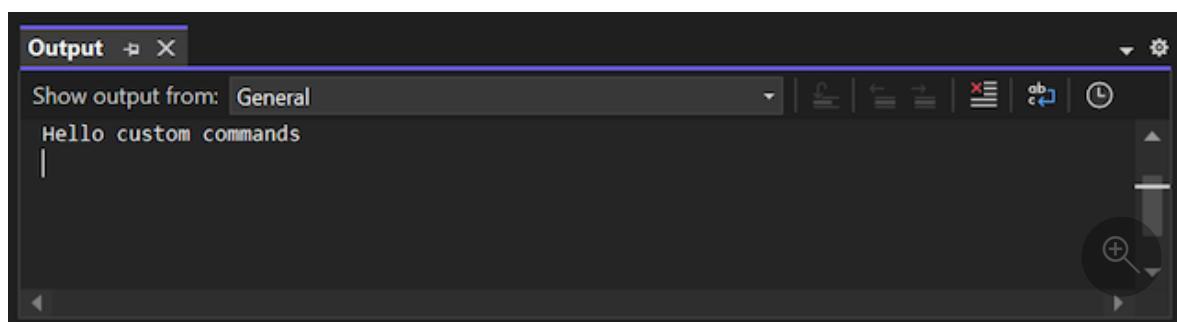
O script de comando personalizado é executado no ambiente ativado para o projeto em Python.

13. Realize a comutação para o editor com o arquivo de projeto. Na definição do elemento `<Target>` (adicionada na [etapa 8](#)), altere o valor do atributo `ExecuteIn` para `output`.

XML

```
<CreatePythonCommandItem  
...  
ExecuteIn="output"  
...  
</CreatePythonCommandItem>
```

14. Salve as alterações, retorne para o Visual Studio e recarregue o projeto.
15. Selecione o comando personalizado **Executar arquivo de inicialização** novamente usando o menu de contexto do **Python**. Agora, a saída do programa aparecerá na janela de **Saída** do Visual Studio em vez de em uma janela do console:



16. Para adicionar mais comandos personalizados, siga este mesmo processo:
 - a. Defina um elemento `<Target>` adequado para o comando personalizado no arquivo de projeto.
 - b. Adicione o valor de atributo `Name` para o elemento `<Target>` no grupo de propriedades `<PythonCommands>`.
 - c. Salve as alterações no arquivo de projeto.
 - d. Recarregue o projeto no Visual Studio.

Usar as propriedades do projeto

Para conferir as propriedades do projeto ou as variáveis de ambiente nos valores de atributos do elemento `<Target>`, use o nome da propriedade em um token `$()`, como `$(StartupFile)` e `$(MSBuildProjectDirectory)`. Para obter mais informações, confira [Propriedades do MSBuild](#).

Se você invocar um comando, como `($StartupFile)`, que usa propriedades do projeto, como a propriedade **StartupFile**, e o comando apresentar falhas porque o token é indefinido, o Visual Studio desabilitará o comando até que você recarregue o projeto. Se você fizer alterações no projeto que modifiquem a definição de propriedade, as

alterações não atualizarão o estado do comando relacionado. Nesse caso, você ainda precisará recarregar o projeto.

Compreender a estrutura do elemento <Target>

Você define os detalhes de um comando personalizado ao usar o elemento <Target>. O formato geral do elemento <Target> é mostrado no pseudocódigo a seguir:

```
XML

<Target Name="Name1" Label="Display Name" Returns="@Commands">
  <CreatePythonCommandItem Target="filename, module name, or code"
    TargetType="executable/script/module/code/pip"
    Arguments="..."
    ExecuteIn="console/consolepause/output/repl[:Display name]/none"
    WorkingDirectory="..."
    ErrorRegex="..."
    WarningRegex="..."
    RequiredPackages="...;..."
    Environment="...">

  <!-- Output always appears in this form, with these exact attributes --
  ->
  <Output TaskParameter="Command" ItemName="Commands" />
</CreatePythonCommandItem>
</Target>
```

Atributos de destino

A tabela apresentada a seguir lista os atributos do elemento <Target>.

[] Expandir a tabela

| Atributo | Obrigatório | Descrição |
|----------|-------------|---|
| Name | Sim | O identificador do comando dentro do projeto do Visual Studio. Esse nome deve ser adicionado ao grupo de propriedades <PythonCommands> para que o comando seja exibido no menu de contexto do Python. |
| Label | Yes | O nome de exibição da interface do usuário que aparece no menu de contexto do Python. |
| Returns | Yes | As informações retornadas, que devem conter o token @(Commands), identificam o destino como um comando. |

Atributos de CreatePythonCommandItem

O elemento `<Target>` contém os elementos `<CreatePythonCommandItem>` e `<Output>`, que definem o comportamento do comando personalizado em detalhes. A tabela apresentada a seguir lista os atributos do elemento `<CreatePythonCommandItem>` disponíveis. Todos os valores de atributo não diferenciam maiúsculas de minúsculas.

 Expandir a tabela

| Attribute | Obrigatório | Descrição |
|-------------------------|-------------|--|
| <code>TargetType</code> | Sim | Especifica o que o atributo <code>Target</code> contém e como o valor é usado em conjunto com o atributo <code>Arguments</code> : - <code>executable</code> : realiza a execução do executável nomeado no atributo <code>Target</code> ao acrescentar o valor no atributo <code>Arguments</code> , como se tivesse sido inserido diretamente na linha de comando. O valor deve conter apenas um nome de programa sem argumentos. - <code>script</code> : realiza a execução do comando <code>python.exe</code> com o nome do arquivo no atributo <code>Target</code> , seguido do valor no atributo <code>Arguments</code> . - <code>module</code> : realiza a execução do comando <code>python -m</code> , seguido do nome do módulo no atributo <code>Target</code> e seguido do valor no atributo <code>Arguments</code> . - <code>code</code> : realiza a execução do código embutido contido no atributo <code>Target</code> . Ignore o valor de atributo <code>Arguments</code> . - <code>pip</code> : realiza a execução do pip com o comando no atributo <code>Target</code> , seguido do valor no atributo <code>Arguments</code> . Se o atributo <code>ExecuteIn</code> estiver definido como <code>output</code> , o pip assumirá que a solicitação é para realizar a execução do comando <code>install</code> e usará o atributo <code>Target</code> como o nome do pacote. |
| <code>Target</code> | Yes | Especifica o nome do arquivo, nome do módulo, código ou comando do pip a ser usado, com base no valor do atributo <code>TargetType</code> . |
| <code>Arguments</code> | Opcional | Fornece uma sequência de argumentos (se houver) para usar com o atributo <code>Target</code> . - Quando o valor de atributo <code>TargetType</code> é <code>script</code> , o valor <code>Arguments</code> é fornecido ao programa em Python em vez de usar o comando <code>python.exe</code> . - Quando o valor de atributo <code>TargetType</code> é <code>code</code> , o valor <code>Arguments</code> é ignorado. |
| <code>ExecuteIn</code> | Yes | Especifica o ambiente no qual a execução do comando deve ser realizada: - <code>console</code> : (padrão) realize a execução do atributo <code>Target</code> com |

| Attribute | Obrigatório | Descrição |
|--|-------------|---|
| | | <p>o valor <code>Arguments</code> como se eles tivessem sido inseridos diretamente na linha de comando. Enquanto o atributo <code>Target</code> está em execução, uma janela de comando é exibida e fechada automaticamente.</p> <ul style="list-style-type: none"> - <code>consolepause</code>: tem um comportamento semelhante ao do <code>console</code>, mas aguarda que uma tecla seja pressionada antes de fechar a janela. - <code>output</code>: realiza a execução do atributo <code>Target</code> e a exibição dos resultados na janela de Saída do Visual Studio. Se o atributo <code>TargetType</code> for <code>pip</code>, o Visual Studio usará o atributo <code>Target</code> como o nome do pacote e acrescentará o valor de atributo <code>Arguments</code>. - <code>rep1</code>: realiza a execução do atributo <code>Target</code> na Janela Interativa do Python. O nome de exibição opcional é usado para o título da janela. - <code>none</code>: tem um comportamento semelhante ao do <code>console</code>. |
| <code>WorkingDirectory</code> | Opcional | Identifica a pasta na qual a execução do comando deve ser realizada. |
| <code>ErrorRegex</code> <code>WarningRegEx</code> | Opcional | Usado somente quando o atributo <code>ExecuteIn</code> está definido como <code>output</code> . Ambos os valores de atributos especificam uma expressão regular que o Visual Studio usa para analisar a saída do comando e mostrar erros e avisos na janela Lista de Erros . Se esses atributos não forem especificados, o comando não afetará a janela Lista de Erros . Para obter mais informações sobre o que o Visual Studio espera, confira Grupos de captura nomeados . |
| <code>RequiredPackages</code> | Opcional | Fornece uma lista de requisitos de pacote para o comando ao usar o mesmo formato do arquivo requirements.txt (pip.readthedocs.io). Por exemplo, o comando Executar PyLint especifica o formato <code>pylint>=1.0.0</code> . Antes de realizar a execução do comando, o Visual Studio confirma que todos os pacotes da lista estão instalados. O Visual Studio usa o pip para instalar todos os pacotes ausentes. |
| <code>Environment</code> | Opcional | Identifica uma sequência de variáveis de ambiente a serem definidas antes da execução do comando. Cada variável usa o formato <code>\<NAME>=\<VALUE></code> com múltiplas variáveis separadas por ponto e vírgula. Uma variável com vários valores deve estar entre aspas simples ou duplas, como em <code>'NAME=VALUE1;VALUE2'</code> . |

Grupos de captura nomeados para expressões regulares

Quando o Visual Studio analisa erros e avisos da saída de comando personalizado, espera-se que as expressões regulares nos valores de atributos `ErrorRegex` e `WarningRegex` usem os seguintes grupos nomeados:

- `(?<message>...)`: texto do erro.
- `(?<code>...)`: valor do código de erro.
- `(?<filename>...)`: nome do arquivo para o qual o erro é relatado.
- `(?<line>...)`: número de linha da localização no arquivo para a qual o erro é relatado.
- `(?<column>...)`: número da coluna da localização no arquivo para a qual o erro é relatado.

Por exemplo, PyLint gera avisos da seguinte forma:

```
Saída

***** Module hello
C: 1, 0: Missing module docstring (missing-docstring)
```

Para permitir que o Visual Studio extraia as informações corretas desses avisos e os mostre na janela **Lista de Erros**, o valor de atributo `WarningRegex` para o comando **Executar PyLint** deve ser semelhante ao seguinte:

```
XML

^(?<filename>.+?)\((?<line>\d+),(?<column>\d+)\): warning (?<msg_id>.+?): (?<message>.+?)$]]
```

ⓘ Observação

A sintaxe `msg_id` no valor de atributo `WarningRegex` deve, na verdade, ser `code`, conforme descrito em [Issue 3680](#).

Importar comandos personalizados ao usar o arquivo de destino

Se você definir comandos personalizados em um arquivo de projeto em Python, os comandos estarão disponíveis somente para esse projeto específico. Quando desejar criar comandos personalizados e usá-los em vários projetos, é possível definir o grupo de propriedades `<PythonCommands>` com todos os seus elementos `<Target>` em um

arquivo de destino (*.targets*) e, em seguida, importar esse arquivo para os projetos em Python.

- O arquivo de destino usa um formato e uma sintaxe semelhantes para definir comandos personalizados, conforme descrito para o arquivo de projeto em Python (*.pyproj*). Os elementos comuns a serem configurados incluem `<PythonCommands>`, `<Target>`, `<CreatePythonCommandItem>` e `<Output>`:

XML

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <PythonCommands>
      $(PythonCommands);
      <!-- Additional command names -->
    </PythonCommands>
  </PropertyGroup>

  <Target Name="..." Label="..." Returns="@.Commands">
    <!-- CreatePythonCommandItem and Output elements... -->
  </Target>

  <!-- Any number of additional Target elements-->
</Project>
```

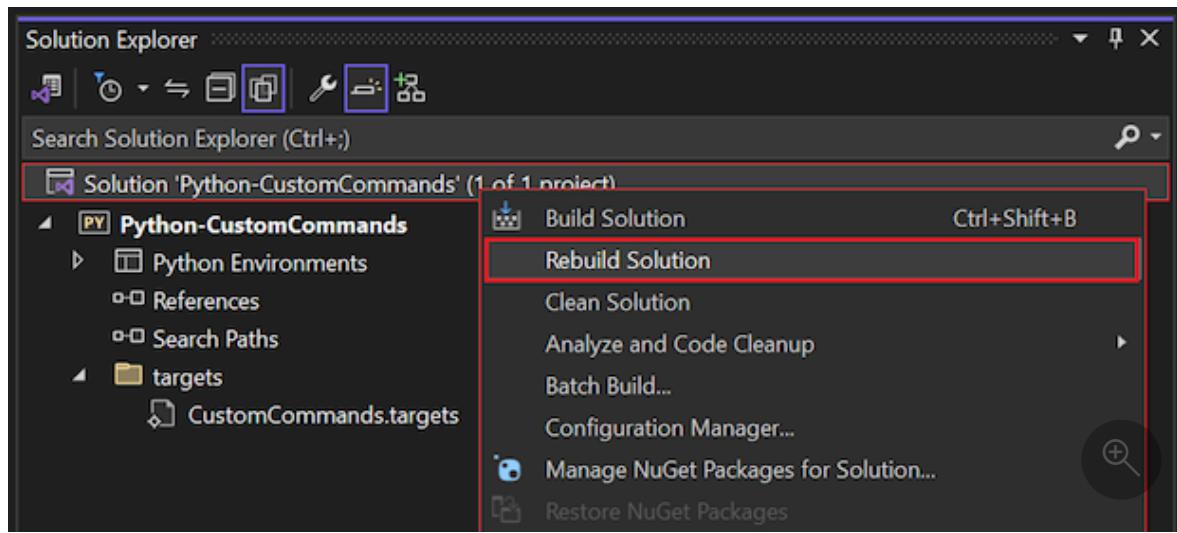
- Para importar um arquivo de destino para o projeto, adicione um elemento `<Import Project="(path)">` em qualquer lugar no elemento `<Project>` no arquivo de projeto.

Por exemplo, se você tiver um arquivo de projeto chamado *CustomCommands.targets* em uma pasta de *destinos* do projeto em Python, adicione o seguinte código ao arquivo de projeto:

XML

```
<Import Project="targets/CustomCommands.targets"/>
```

- Se o arquivo de projeto importar um arquivo de destino e você fizer alterações no arquivo de destino enquanto o projeto estiver aberto no Visual Studio, será necessário **Recompilar a solução** do Visual Studio que contém o projeto, e não apenas o projeto.



Comandos de exemplo

As seções apresentadas a seguir fornecem um código de exemplo que você pode usar para definir comandos personalizados para os projetos em Python.

Executar PyLint (destino do módulo)

O seguinte código é exibido no arquivo *Microsoft.PythonTools.targets*:

```
XML

<PropertyGroup>
    <PythonCommands>$(PythonCommands);PythonRunPyLintCommand</PythonCommands>
    <PyLintWarningRegex>
        <![CDATA[^(?<filename>.+?)\((?<line>\d+),(?<column>\d+)\): warning (?<msg_id>.+?): (?<message>.+?)$]]>
    </PyLintWarningRegex>
</PropertyGroup>

<Target Name="PythonRunPyLintCommand"
    Label="resource:Microsoft.PythonTools.Common;Microsoft.PythonTools.Common.Strings;RunPyLintLabel"
    Returns="@(<Commands>)">
    <CreatePythonCommandItem Target="pylint.lint"
        TargetType="module"
        Arguments="\"--msg-template={abspath}({line},{column}): warning {msg_id}: {msg} [{C}:{symbol}]\" -r n @(<Compile,>'')
        WorkingDirectory="$(MSBuildProjectDirectory)"
        ExecuteIn="output"
        RequiredPackages="pylint>=1.0.0"
        WarningRegex="$(PyLintWarningRegex)">
    <Output TaskParameter="Command" ItemName="Commands" />

```

```
</CreatePythonCommandItem>
</Target>
```

Executar pip install com um pacote específico (pip target)

O comando apresentado a seguir realiza a execução do comando `pip install my-package` na janela de Saída do Visual Studio. É possível usar um comando como este ao desenvolver um pacote e testar a instalação dele. O elemento `<Target>` contém o nome do pacote em vez do comando `install`, que é assumido quando você usa a definição do atributo `ExecuteIn="output"`.

XML

```
<PropertyGroup>
  <PythonCommands>$(PythonCommands);InstallMyPackage</PythonCommands>
</PropertyGroup>

<Target Name="InstallMyPackage" Label="pip install my-package"
Returns="@.Commands">
  <CreatePythonCommandItem Target="my-package" TargetType="pip" Arguments=""
    WorkingDirectory="$(MSBuildProjectDirectory)" ExecuteIn="output">
    <Output TaskParameter="Command" ItemName="Commands" />
  </CreatePythonCommandItem>
</Target>
```

Mostrar pacotes de pip desatualizados (destino de pip)

O comando apresentado a seguir executa o pip com a função `list` para identificar pacotes desatualizados do pip:

XML

```
<PropertyGroup>
  <PythonCommands>$(PythonCommands);ShowOutdatedPackages</PythonCommands>
</PropertyGroup>

<Target Name="ShowOutdatedPackages" Label="Show outdated pip packages"
Returns="@.Commands">
  <CreatePythonCommandItem Target="list" TargetType="pip" Arguments="-o --
format columns"
    WorkingDirectory="$(MSBuildProjectDirectory)" ExecuteIn="consolepause">
    <Output TaskParameter="Command" ItemName="Commands" />
  </CreatePythonCommandItem>
</Target>
```

Executar o executável com consolepause

O comando apresentado a seguir realiza a execução da função `where` para mostrar a localização dos arquivos em Python começando pela pasta do projeto:

XML

```
<PropertyGroup>

<PythonCommands>$(PythonCommands);ShowAllPythonFilesInProject</PythonCommand
s>
</PropertyGroup>

<Target Name="ShowAllPythonFilesInProject" Label="Show Python files in
project" Returns="@((Commands))">
  <CreatePythonCommandItem Target="where" TargetType="executable"
Arguments="/r . *.py"
    WorkingDirectory="$(MSBuildProjectDirectory)" ExecuteIn="output">
    <Output TaskParameter="Command" ItemName="Commands" />
  </CreatePythonCommandItem>
</Target>
```

Comandos para executar o servidor e executar o servidor de depuração

Para explorar como os comandos **Iniciar servidor** e **Iniciar servidor de depuração** para projetos Web são definidos, examine o repositório [Microsoft.PythonTools.Web.targets](#) no GitHub.

Instalar o pacote para desenvolvimento

O código apresentado a seguir realiza a execução do pip para instalar pacotes:

XML

```
<PropertyGroup>
  <PythonCommands>PipInstallDevCommand;$(PythonCommands);</PythonCommands>
</PropertyGroup>

<Target Name="PipInstallDevCommand" Label="Install package for development"
Returns="@((Commands))">
  <CreatePythonCommandItem Target="pip" TargetType="module"
Arguments="install --editable $(ProjectDir)"
    WorkingDirectory="$(WorkingDirectory)" ExecuteIn="Repl:Install
package for development">
    <Output TaskParameter="Command" ItemName="Commands" />
```

```
</CreatePythonCommandItem>
</Target>
```

De [fxthomas/Example.pyproj.xml](https://github.com/fxthomas/Example.pyproj.xml) (GitHub), usado com permissão.

Gerar o Windows Installer

O script apresentado a seguir gera um Windows Installer:

XML

```
<PropertyGroup>
  <PythonCommands>$({PythonCommands});BdistWinInstCommand;</PythonCommands>
</PropertyGroup>

<Target Name="BdistWinInstCommand" Label="Generate Windows Installer"
Returns="@({Commands})">
  <CreatePythonCommandItem Target="$(ProjectDir)setup.py"
TargetType="script"
    Arguments="bdist_wininst --user-access-control=force --title
&quot;${InstallerTitle}&quot; --dist-
dir=&quot;${DistributionOutputDir}&quot;"
    WorkingDirectory="$(WorkingDirectory)" RequiredPackages="setuptools"
    ExecuteIn="Repl:Generate Windows Installer">
    <Output TaskParameter="Command" ItemName="Commands" />
  </CreatePythonCommandItem>
</Target>
```

De [fxthomas/Example.pyproj.xml](https://github.com/fxthomas/Example.pyproj.xml) (GitHub), usado com permissão.

Gerar pacote wheel para o Python

O script apresentado a seguir gera um pacote [wheel para o Python](#):

XML

```
<PropertyGroup>
  <PythonCommands>$({PythonCommands});BdistWheelCommand;</PythonCommands>
</PropertyGroup>

<Target Name="BdistWheelCommand" Label="Generate Wheel Package"
Returns="@({Commands})">
  <CreatePythonCommandItem Target="$(ProjectDir)setup.py"
TargetType="script"
    Arguments="bdist_wheel --dist-
dir=&quot;${DistributionOutputDir}&quot;"
    WorkingDirectory="$(WorkingDirectory)"
    RequiredPackages="wheel;setuptools"
```

```
    ExecuteIn="Repl:Generate Wheel Package">
    <Output TaskParameter="Command" ItemName="Commands" />
  </CreatePythonCommandItem>
</Target>
```

De [fxthomas/Example.pyproj.xml](#) (GitHub), usado com permissão.

Solucionar problemas de comandos personalizados

Faça uma revisão das seções apresentadas a seguir sobre possíveis problemas relacionados ao trabalho com comandos personalizados.

Arquivo de projeto não carregado

Essa mensagem de erro indica que há erros de sintaxe no arquivo de projeto. A mensagem inclui o erro específico com um número de linha e a posição do caractere.

A janela do console fecha após a execução do comando

Se a janela do console fechar imediatamente após a execução do comando, use a definição de atributo `ExecuteIn="consolepause"` em vez de `ExecuteIn="console"`.

Comando ausente no menu

Se você não vir o comando personalizado no menu de contexto do Python, verifique os seguintes itens:

- Confirme se o comando está incluído no grupo de propriedades `<PythonCommands>`.
- Verifique se o nome do comando, conforme definido na lista de comandos, corresponde ao nome especificado no elemento `<Target>`.

Veja um exemplo. No trecho em XML apresentado a seguir, o nome `Example` no grupo de propriedades `<PythonCommands>` não corresponde ao nome `ExampleCommand` na definição do elemento `<Target>`. O Visual Studio não encontra um comando chamado `Example`, portanto, nenhum comando é exibido. Use `ExampleCommand` na lista de comandos ou altere o nome do destino para ser apenas `Example`.

XML

```
<PropertyGroup>
  <PythonCommands>$({PythonCommands});Example</PythonCommands>
</PropertyGroup>
<Target Name="ExampleCommand" Label="Example Command"
Returns="@({Commands})">
  <!-- ... -->
</Target>
```

Erro ao executar comando e falha ao obter o destino do comando

Essa mensagem de erro indica que o conteúdo dos elementos `<Target>` ou `<CreatePythonCommandItem>` está incorreto.

Confira a seguir alguns possíveis motivos para esse erro:

- O atributo do elemento `<Target>` necessário está vazio.
- O atributo `TargetType` obrigatório está vazio ou contém um valor não reconhecido.
- O atributo `ExecuteIn` obrigatório está vazio ou contém um valor não reconhecido.
- O atributo `ErrorRegex` ou `WarningRegex` é especificado sem definir a definição de atributo `ExecuteIn="output"`.
- Atributos não reconhecidos existem no elemento. Por exemplo, a referência do atributo pode estar escrita incorretamente como `Argumnets` em vez de `Arguments`.

Os valores de atributos poderão ficar vazios se você fizer referência a uma propriedade indefinida. Se você usar o token `$(StartupFile)`, mas nenhum arquivo de inicialização estiver definido no projeto, o token será resolvido como uma sequência vazia. Nesses casos, talvez você queira definir um valor padrão. Por exemplo, os comandos **Executar servidor** e **Executar servidor de depuração** definidos nos modelos de projeto em Bottle, em Flask e em Django usam o arquivo *manage.py* por padrão (se um arquivo de inicialização do servidor não for especificado nas propriedades do projeto).

O Visual Studio para de responder ou falha

Se o Visual Studio parar de responder e falhar quando você realiza a execução do comando personalizado, provavelmente você está tentando executar um comando de console com a definição de atributo `ExecuteIn="output"`. Nesses casos, o Visual Studio pode falhar ao tentar analisar a saída. Para evitar essa condição, use a definição de atributo `ExecuteIn="console"` em vez disso. Para obter mais informações, confira [Issue 3681](#).

Comando não reconhecido como programa operável ou arquivo em lotes

Ao definir a definição do atributo `TargetType="executable"`, o valor no atributo `Target` deve ser *somente* o nome do programa sem quaisquer argumentos, como apenas `python` ou `python.exe`. Nesse caso, mova os argumentos para o atributo `Arguments`.

Comentários

Esta página foi útil?

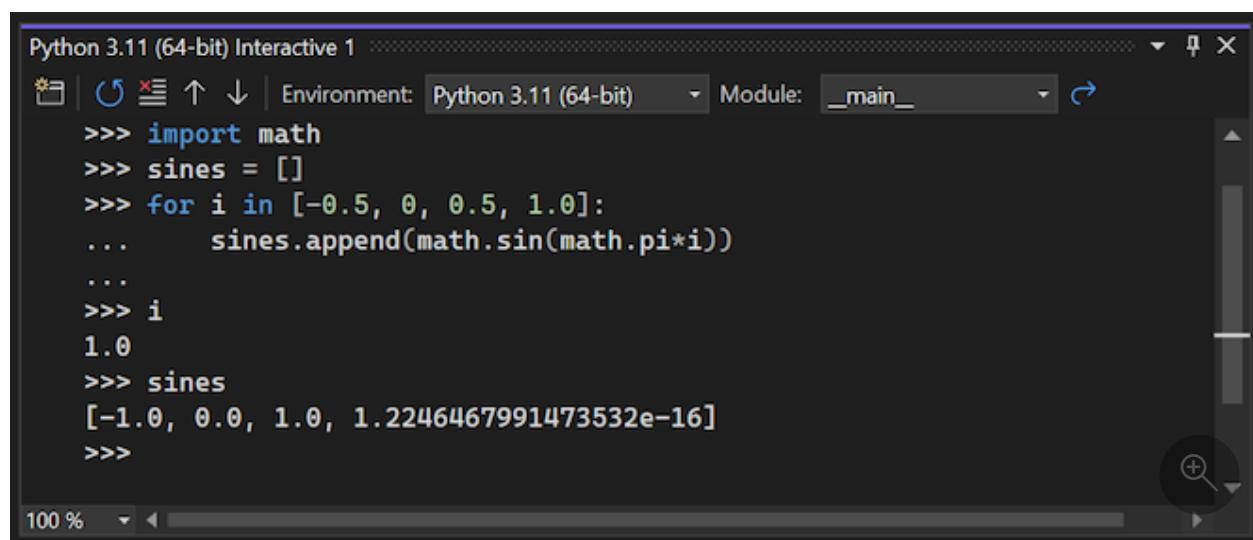
 Yes

 No

Trabalho com a Janela Interativa do Python no Visual Studio

Artigo • 23/04/2024

O Visual Studio fornece uma janela de Loop de Leitura, Avaliação e Impressão (REPL) interativa para cada um de seus ambientes do Python, que melhora o REPL obtido com o comando `python.exe` na linha de comando. A **Janela Interativa do Python** permite inserir código Python arbitrário e visualizar resultados de forma imediata. Essa abordagem de codificação ajuda você a aprender e experimentar APIs e bibliotecas e a desenvolver de maneira interativa um código funcional para incluir em projetos.



The screenshot shows the Python 3.11 (64-bit) Interactive window in Visual Studio. The window title is "Python 3.11 (64-bit) Interactive 1". The environment dropdown shows "Python 3.11 (64-bit)" and the module dropdown shows "__main__". The code input area contains:

```
>>> import math
>>> sines = []
>>> for i in [-0.5, 0, 0.5, 1.0]:
...     sines.append(math.sin(math.pi*i))
...
>>> i
1.0
>>> sines
[-1.0, 0.0, 1.0, 1.2246467991473532e-16]
>>>
```

The output shows the value of `i` as 1.0 and the resulting list `sines` containing [-1.0, 0.0, 1.0, 1.2246467991473532e-16]. The bottom right corner of the window has a magnifying glass icon with a plus sign.

O Visual Studio tem diversos modos de REPL do Python à sua disposição:

[\[+\] Expandir a tabela](#)

| REPL | Descrição | Edição | Depuração | Imagens |
|----------|--|---|--|-----------------------|
| Standard | O REPL padrão, que se comunica com o Python diretamente | Edição Standard (com múltiplas linhas e mais) | Sim, por meio de <code>\$attach</code> | Não |
| Depurar | REPL padrão, que se comunica com o processo depurado do Python | Edição padrão | Somente depuração | Não |
| IPython | O REPL se comunica com o back-end do IPython | Comandos do IPython, funcionalidades do Pylab | Não | Sim, embutido no REPL |

| REPL | Descrição | Edição | Depuração | Imagens |
|-------------------|--|----------------|-----------|-----------------------------|
| IPython sem Pylab | O REPL se comunica com o back-end do IPython | IPython padrão | Não | Sim, em uma janela separada |

Este artigo descreve os modos **REPL Padrão** e **Depuração**. Para obter detalhes sobre os modos do IPython, confira [Usar o REPL do IPython](#).

Para obter um passo a passo em detalhes com exemplos, incluindo as interações com o editor, como **Ctrl+Enter**, confira [Usar a janela Interativa REPL](#).

Pré-requisitos

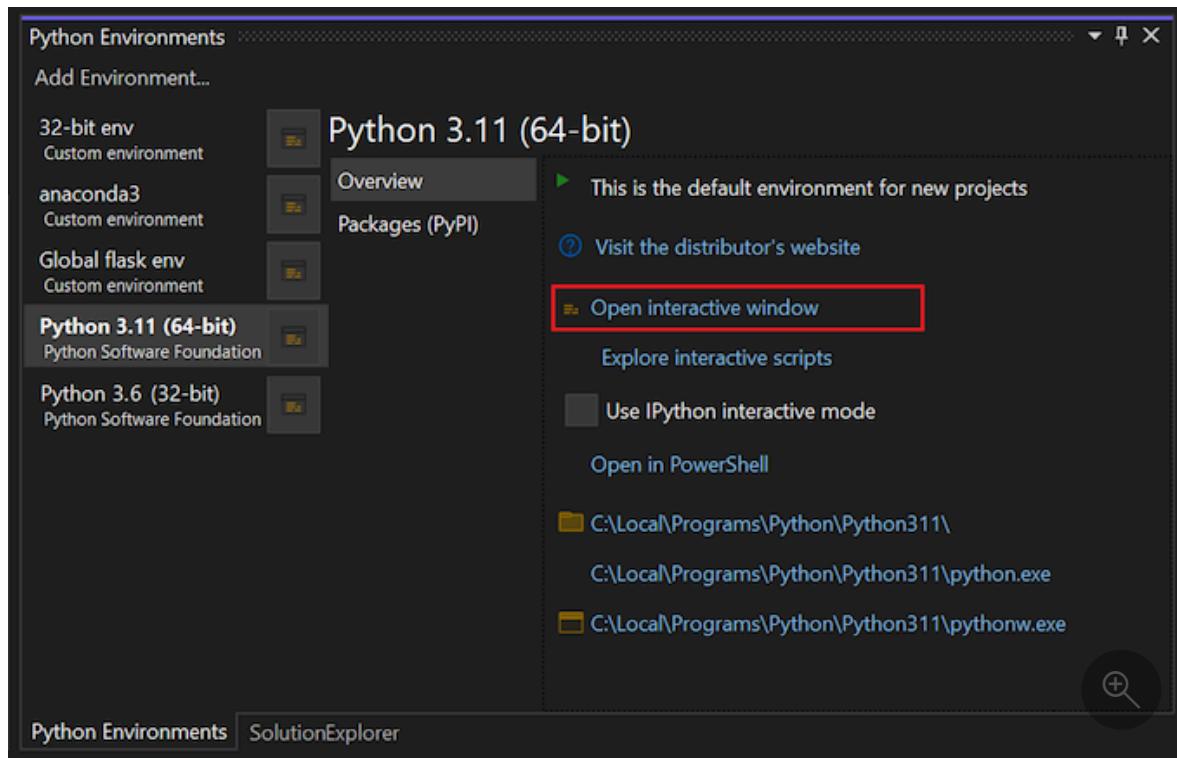
- Instalação do Visual Studio no Windows com suporte para cargas de trabalho em Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Ainda não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

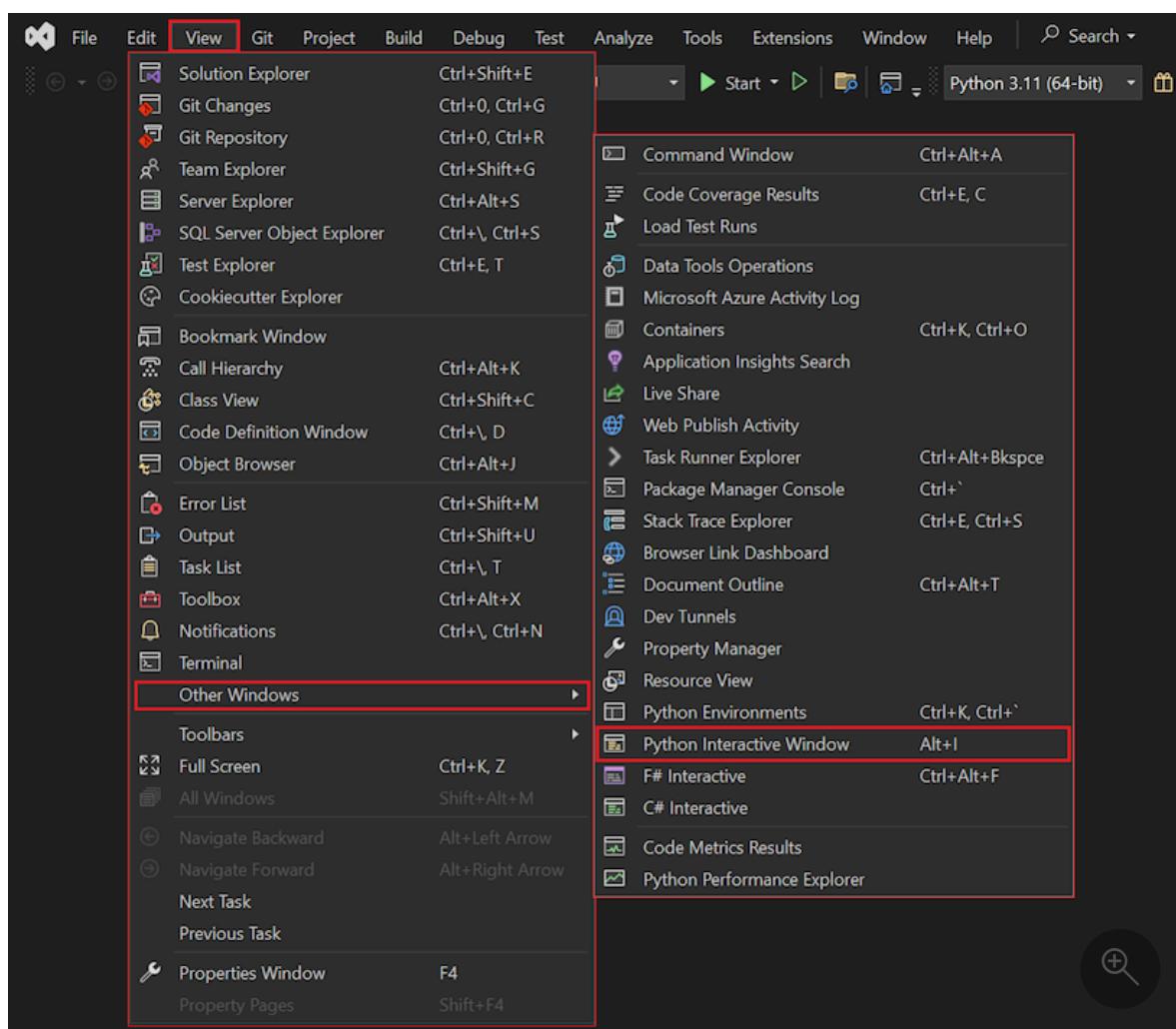
Abrir a Janela Interativa

Existem várias maneiras de abrir a **Janela Interativa** para um ambiente do Python.

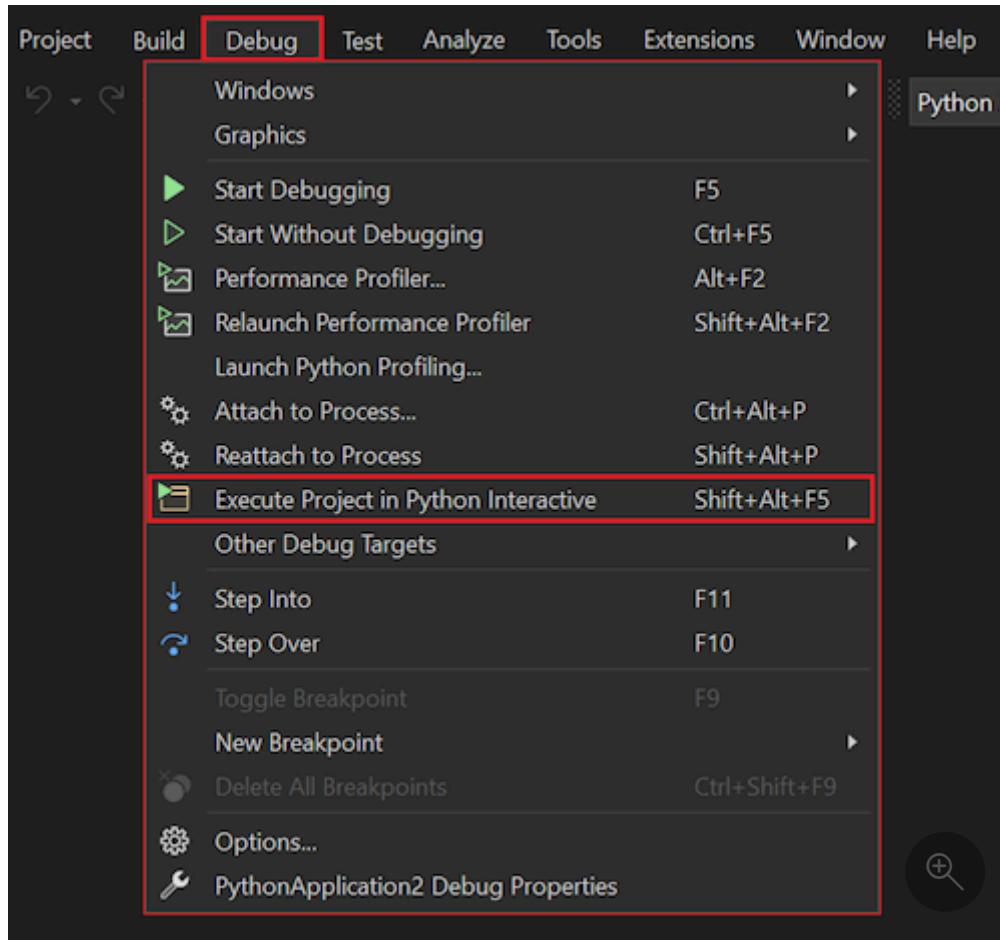
- Na janela **Ambientes do Python**:
 - Selecione **Exibir>Outras Janelas>Ambientes do Python** para abrir a janela **Ambientes do Python** (ou use o atalho de teclado **Ctrl+K>Ctrl+`**).
 - Na janela **Ambientes do Python**, selecione um ambiente e realize a comutação para a página de **Visão Geral** do ambiente.
 - Na página de **Visão Geral**, selecione a opção **Abrir Janela Interativa**.



- No menu **Exibir** na ferramenta do Visual Studio, selecione **Outras Janelas>Janela Interativa do Python:**



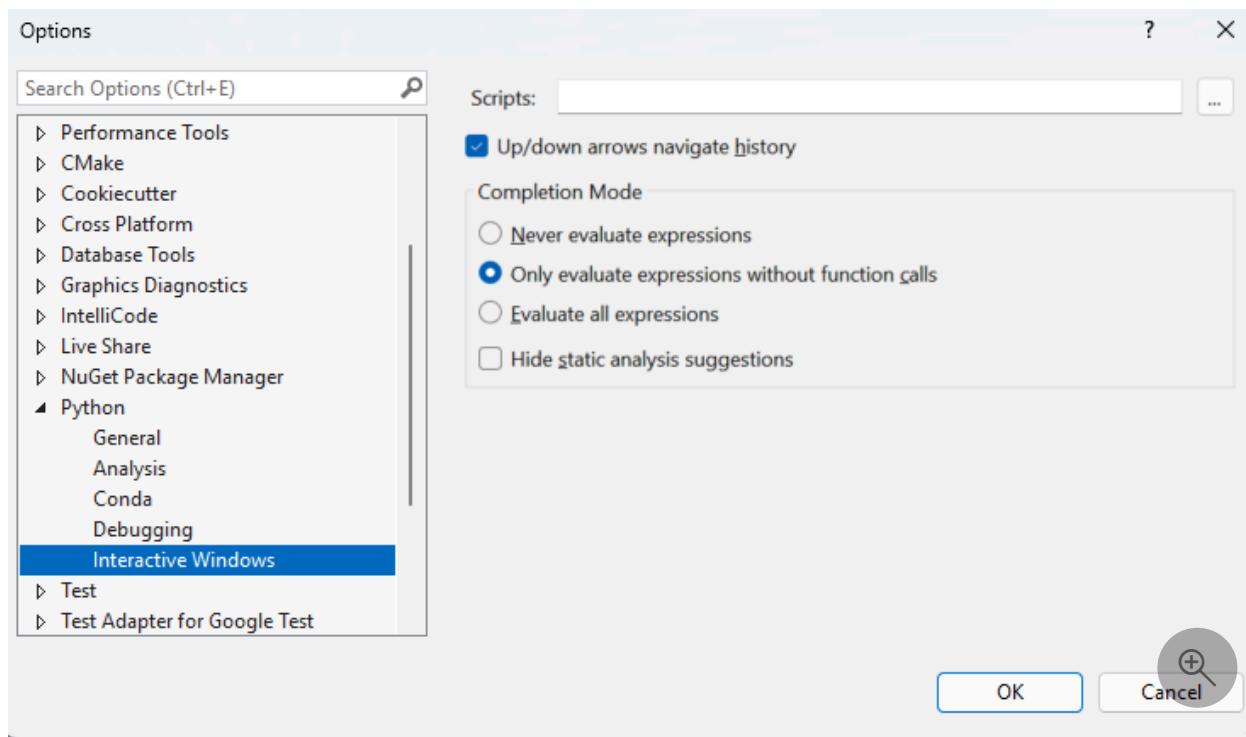
- No menu **Depurar** na barra de ferramentas do Visual Studio, selecione **Executar <Projeto | Arquivo> em Interativo do Python** ou use o atalho de teclado **Shift+Alt+F5**. É possível abrir uma **Janela Interativa** para o **Arquivo de Inicialização** no projeto ou para arquivos independentes:



Outra opção é enviar o código que você está gravando no editor do Visual Studio para a **Janela Interativa**. Essa abordagem é descrita em [Enviar código para a Janela Interativa](#).

Explorar as opções da Janela Interativa

É possível obter o controle sobre vários aspectos da **Janela Interativa**, como a localização dos arquivos de script de inicialização e como as teclas de direção funcionam no ambiente da janela. Para obter acesso às opções, selecione **Ferramentas>Opções>Python>Janelas Interativas**:



As opções são descritas em detalhes em [Python Interactive Window options for Visual Studio](#).

Usar a Janela Interativa

Na **Janela Interativa**, é possível começar a inserir o código linha por linha no prompt de comando `>>>` de REPL do Python. À medida que você inserir cada linha, o Visual Studio realizará a execução do código, incluindo a importação dos módulos necessários e a definição de variáveis.

Quando o Visual Studio detectar que uma linha de código não realiza a formação de uma instrução completa, a solicitação de código sofre alteração para a continuação de REPL `....`. Essa solicitação indica que você precisa inserir mais linhas de código para concluir o bloco de instruções. O Visual Studio aguarda a sintaxe de fechamento antes de tentar executar o bloco de código.

Quando você definir uma instrução `for`, a primeira linha de código iniciará o bloco `for` e terminará com dois-pontos. O bloco pode consistir em uma ou mais linhas de código que definem as tarefas a serem concluídas durante o loop `for`. Ao selecionar **Enter** em uma linha em branco, a **Janela Interativa** fecha o bloco e o Visual Studio executa o código.

Comparação do REPL da linha de comando

A **Janela Interativa** aprimora a experiência do REPL usual da linha de comando do Python ao recuar automaticamente as instruções que pertencem a um escopo circundante. Ambas as abordagens permitem usar as teclas de direção para navegar pelo código inserido. A **Janela Interativa** também fornece itens com múltiplas linhas, enquanto o REPL da linha de comando fornece somente linhas únicas.

Comando meta

A **Janela Interativa** oferece suporte a diversos comandos meta. Todos os comandos meta começam com o símbolo do cifrão `$`. É possível inserir `$help` para visualizar uma lista de comandos meta e `$help <command>` para obter detalhes de uso de um comando específico. A tabela apresentada a seguir resume os comandos meta.

 Expandir a tabela

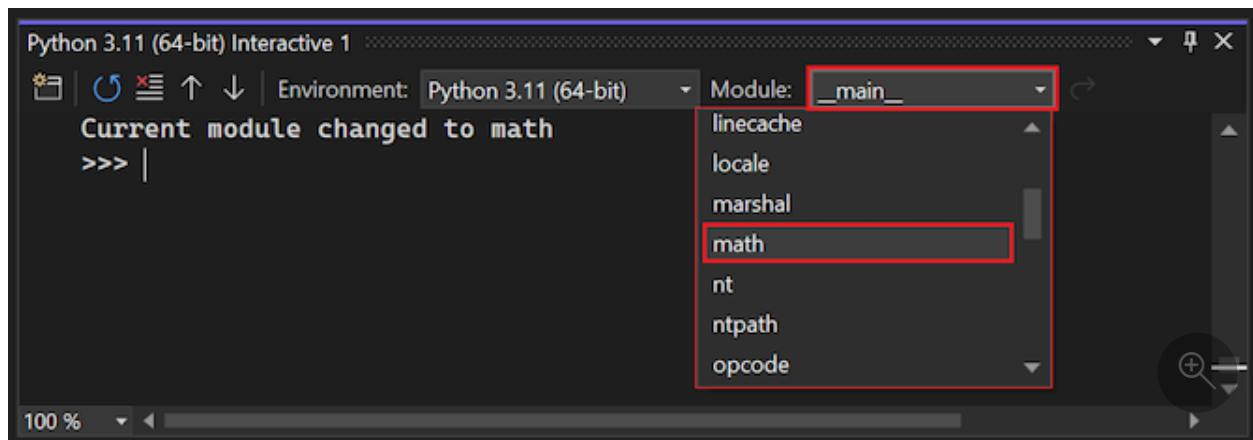
| Metacomando | Descrição |
|---|--|
| <code>\$\$</code> | Inserção de um comentário, que será útil para deixar comentários sobre o código durante a sessão. |
| <code>\$cls</code> , <code>\$clear</code> | Limpeza do conteúdo da janela do editor, mas o histórico e o contexto de execução permanecem intactos. |
| <code>\$help</code> | Exibe uma lista de comandos ou a ajuda sobre um comando específico. |
| <code>\$load</code> | Carregamento de comandos do arquivo e execução até eles estarem concluídos. |
| <code>\$mod</code> | Comutação do escopo atual para o nome do módulo especificado. |
| <code>\$reset</code> | Restauração do ambiente de execução para o estado inicial, mas o histórico é mantido. |
| <code>\$wait</code> | Espera pelo menos o número especificado de milissegundos. |

Além disso, é possível ampliar os comandos com extensões do Visual Studio ao implementar e exportar a classe `IInteractiveWindowCommand`. Para obter mais informações, faça a revisão de um ([exemplo no GitHub](#)).

Comutar o escopo da Janela Interativa

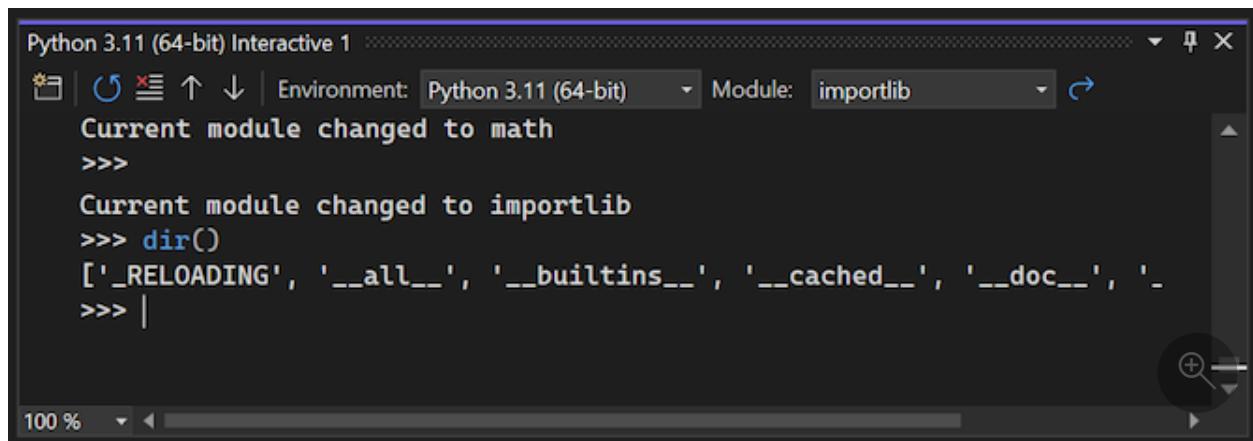
Por padrão, a **Janela Interativa** para um projeto tem como escopo o **Arquivo de Inicialização** do projeto, como se você tivesse executado o arquivo do programa no prompt de comando. Para um arquivo independente, o escopo é definido para esse

arquivo. Durante a sessão do REPL, é possível usar o menu suspenso **Escopo do módulo** para alterar o escopo quando desejar:



Ao importar um módulo, como `import importlib`, opções aparecem no menu suspenso **Escopo do módulo** para a comutação para qualquer escopo nesse módulo. Uma mensagem na **Janela Interativa** reporta as alterações para o novo escopo, para que seja possível para você acompanhar como chegou a um determinado estado durante a sessão ao realizar a revisão do histórico de comando.

Se você inserir o comando `dir()` em um escopo, o Visual Studio exibirá identificadores válidos nesse escopo, incluindo nomes de funções, classes e variáveis. A imagem apresentada a seguir mostra o resultado do comando `dir()` para o escopo `importlib`:

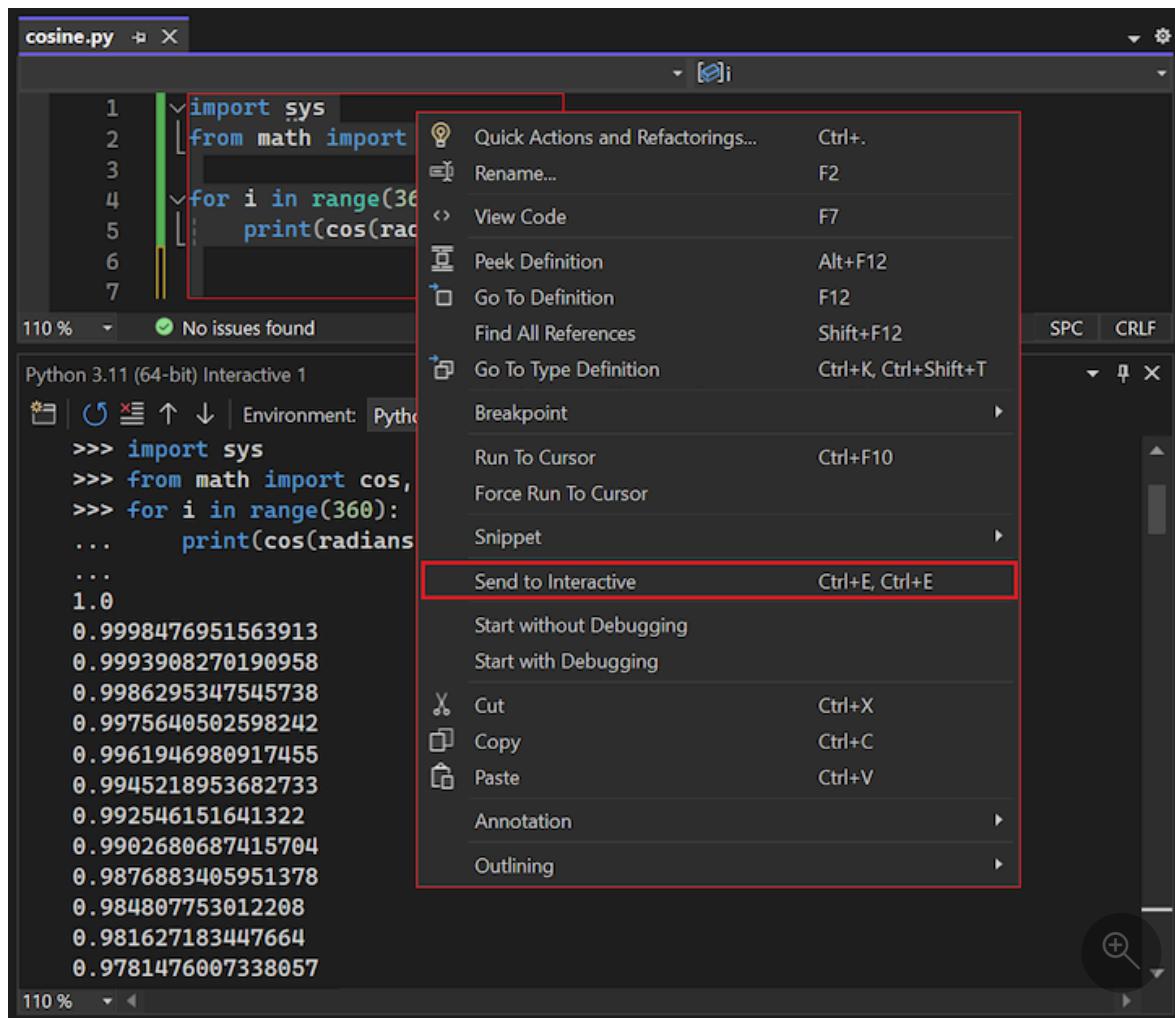


Enviar código para a Janela Interativa

Além de trabalhar diretamente na **Janela Interativa**, é possível enviar código do editor do Visual Studio para a janela. Esse comando é útil para o desenvolvimento de código iterativo ou evolucionário, incluindo o teste do código durante o desenvolvimento.

1. Abra um arquivo de código no editor do Visual Studio e selecione uma parte ou todo o código.

2. Clique com o botão direito do mouse no código selecionado e escolha **Enviar para o Interativo** (ou use o atalho de teclado **Ctrl+E, E**).



Modificação e execução do código

Após enviar o código para a **Janela Interativa** e realizar a exibição da saída, você poderá modificar o código e testar as alterações. Use as teclas de direção para cima e para baixo para rolar até o código no histórico de comando da janela. Realize a modificação do código e a execução do código atualizado ao selecionar **Ctrl+Enter**.

Ao fazer alterações, se você selecionar **Enter** no final de uma instrução de código completa, o Visual Studio executará o código. Se a instrução do código não estiver completa, o Visual Studio inserirá uma nova linha na janela.

Salvar o código e remover as solicitações

Após concluir o trabalho no código, você poderá selecionar o código atualizado na **Janela Interativa** e colá-lo novamente no arquivo de projeto para salvar o trabalho.

Ao colar o código da **Janela Interativa** no editor, o Visual Studio remove o prompt de comando `>>>` do REPL e a solicitação de continuação `...`, por padrão. Esse comportamento permite transferir o código da janela para o editor com facilidade.

É possível alterar o comportamento com a opção **Colar remove as solicitações do REPL para a Janela Interativa**:

1. Selecione **Ferramentas>Opções** para abrir a caixa de diálogo **Opções**.
2. Expanda a seção **Editor de Texto>Python>Formatação**.
3. Desmarque a opção **Colar remove as solicitações do REPL**.

Quando você desmarca a opção, os caracteres da solicitação são retidos no código colado da janela. Para obter mais informações, confira [Opções: Opções diversas](#).

Revisar o comportamento do IntelliSense

A **Janela Interativa** inclui sugestões do IntelliSense com base em objetos ativos, diferentemente do editor de código, no qual o IntelliSense se baseia somente na análise do código-fonte. Como resultado, as sugestões do IntelliSense na **Janela Interativa** são mais corretas, especialmente com um código gerado de forma dinâmica. A desvantagem é que as funções com efeitos colaterais, como mensagens de registro em log, podem afetar a experiência de desenvolvimento.

É possível realizar o ajuste do comportamento do Intellisense ao usar as opções de **Conclusão**:

1. Selecione **Ferramentas>Opções** para abrir a caixa de diálogo **Opções**.
2. Expanda a seção **Python>Janelas Interativas**.
3. Ajuste as configurações no grupo **Modo de Conclusão**, como **Nunca avaliar as expressões** ou **Ocultar as sugestões de análise estática**.

Para obter mais informações, confira [Opções: Opções da Janela Interativa](#).

Conteúdo relacionado

- [Usar o IPython na Janela Interativa no Visual Studio](#)
- [Tutorial: Use the Interactive REPL window in Visual Studio](#)

Comentários

Esta página foi útil?

 Yes

 No

Usar o IPython na Janela Interativa no Visual Studio

Artigo • 23/04/2024

A **Janela Interativa** do Visual Studio no modo IPython é um ambiente de desenvolvimento interativo avançado, porém amigável, que tem recursos de Computação Paralela Interativa. O presente artigo fornece instruções sobre como usar o IPython na **Janela Interativa** do Visual Studio e obter acesso aos recursos padrão da **Janela Interativa**.

Pré-requisitos

- Instalação do Visual Studio no Windows com suporte para cargas de trabalho em Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Ainda não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux funciona perfeitamente com o Python por meio das extensões disponíveis ↗.

- A instalação do Python deve incluir as bibliotecas IPython, numpy e matplotlib. É possível instalar essas bibliotecas ao usar o instalador de Pacote no Visual Studio, conforme descrito em [Tutorial: Install packages in your Python environment in Visual Studio](#).

ⓘ Observação

A implementação IronPython não oferece suporte ao IPython, apesar de ser possível fazer a seleção dessa opção no formulário **Opções Interativas**. Para obter mais informações, confira a [solicitação de recurso \(Support IPython when interp is IronPython\)](#) ↗.

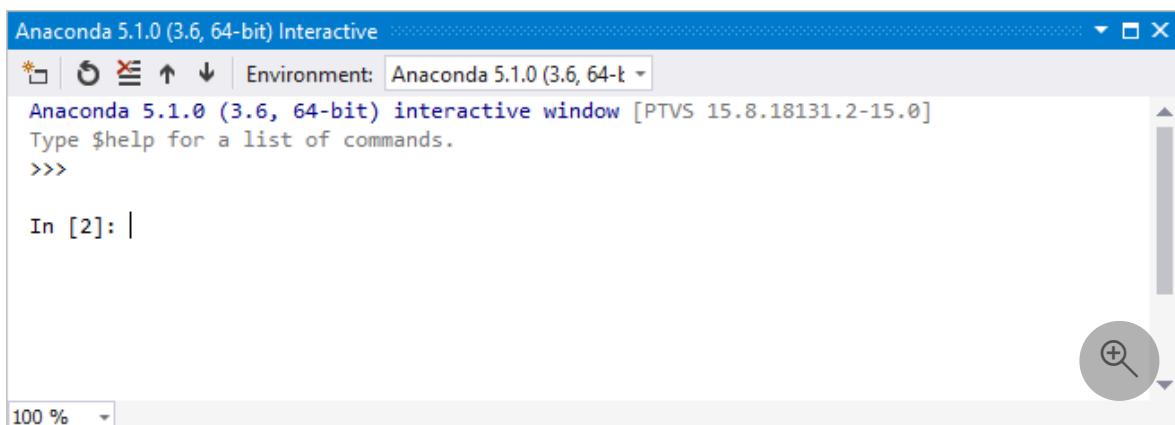
Trabalho com a Janela Interativa

As etapas apresentadas a seguir mostram como usar os comandos do IPython na Janela Interativa. Esse passo a passo pressupõe que você esteja usando o Anaconda.

1. No Visual Studio, selecione **Exibir>Outras Janelas>Ambientes do Python** para abrir a janela **Ambientes do Python**.
2. Na janela **Ambientes do Python**, selecione um ambiente Anaconda.
3. Realize a comutação para a exibição **Pacotes** do ambiente para visualizar os pacotes instalados. No menu suspenso, selecione **Pacotes (Conda)**. A opção de menu pode ser chamada de **pip** ou de **Pacotes**.
4. Confirme se as bibliotecas `ipython` e `matplotlib` estão instaladas para o ambiente.

Se ambas as bibliotecas não estiverem instaladas, siga as instruções em [Tutorial: Install packages in your Python environment in Visual Studio](#). Para obter mais informações, confira a [Guia Pacotes](#) na referência de guias da janela Ambientes do Python.

5. Realize a comutação para a exibição **Visão Geral** do ambiente e selecione a opção **Usar o modo interativo IPython**.
6. Selecione **Abrir janela interativa** para exibir a **Janela Interativa** no modo IPython. Talvez seja necessário restaurar a janela se ela já estiver aberta.
7. Quando a **Janela Interativa** for aberta, você deverá ver a primeira solicitação `In [1]`. Se você vir a solicitação padrão `>>>`, insira “devolver” na solicitação para garantir que a janela esteja usando o modo IPython. A solicitação deve ser alterada para algo semelhante à `In [2]`.



8. Insira o seguinte código:

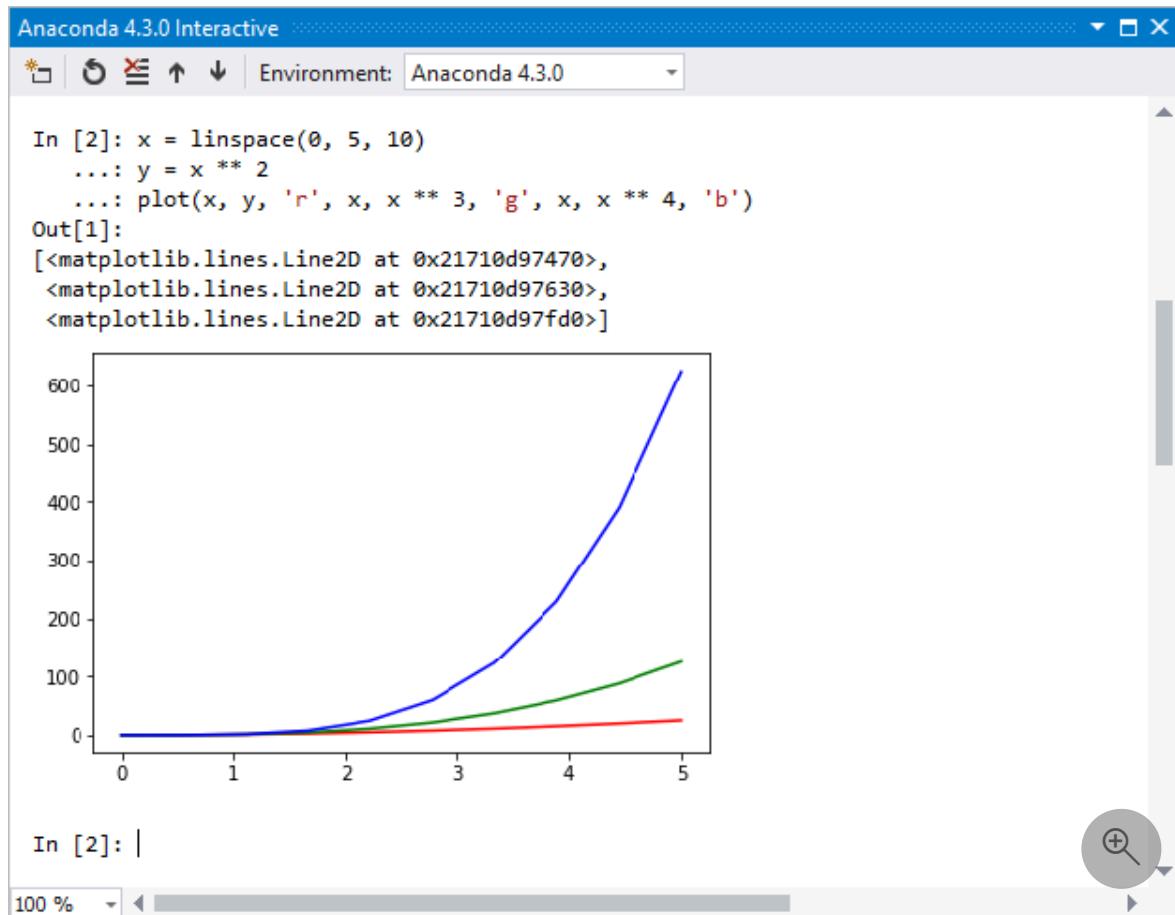
```
Python

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 5, 10)
```

```
y = x ** 2  
plt.plot(x, y, 'r', x, x ** 3, 'g', x, x ** 4, 'b')
```

9. Depois de inserir a última linha de código e selecionar **Enter**, você deverá ver uma representação em gráfico embutida:



É possível redimensionar a representação em gráfico ao arrastá-la no canto inferior direito.

10. Em vez de inserir o código diretamente na **Janela Interativa**, é possível realizar a gravação do código no editor do Visual Studio e enviá-lo para a **Janela Interativa**:

- Cole o código apresentado a seguir em um novo arquivo no editor.
- Use o atalho de teclado **Ctrl+A** para selecionar o código no editor.
- Clique com o botão direito do mouse no código selecionado e escolha **Enviar para o Interativo**. Você também pode usar o atalho de teclado **Ctrl+Enter**.

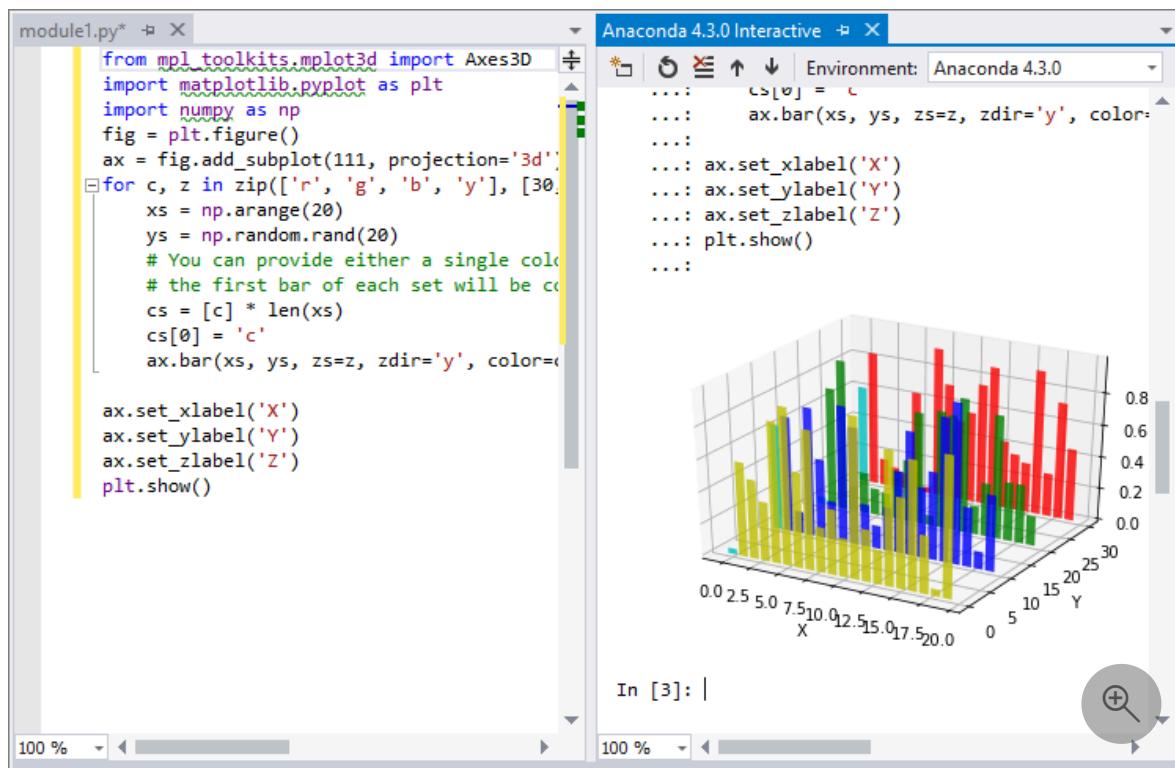
O Visual Studio envia o código como um bloco único para a **Janela Interativa** para evitar uma representação em gráfico intermediária ou parcial.

(Se você não tiver um projeto em Python aberto com um ambiente ativo específico, o Visual Studio abrirá uma **Janela Interativa** para o ambiente padrão listado na janela **Ambientes do Python**.)

Python

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for c, z in zip(['r', 'g', 'b', 'y'], [30, 20, 10, 0]):
    xs = np.arange(20)
    ys = np.random.rand(20)
    # You can provide either a single color or an array. To demonstrate
    # this,
    # the first bar of each set is colored cyan.
    cs = [c] * len(xs)
    cs[0] = 'c'
    ax.bar(xs, ys, zs=z, zdir='y', color=cs, alpha=0.8)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```



11. Para realizar a exibição das representações em gráfico de forma externa à Janela Interativa, execute o código com o comando Depurar>Começar sem a Depuração na barra de ferramentas principal do Visual Studio.

O IPython tem muitos outros recursos úteis, como o escape para o shell do sistema, a substituição de variáveis, a captura de saída, e assim por diante. Para obter mais informações, confira a [documentação do IPython](#).

Conteúdo relacionado

- [Trabalho com a Janela Interativa do Python no Visual Studio](#)
 - [Tutorial: Use the Interactive REPL window in Visual Studio](#)
-

Comentários

Esta página foi útil?

 Yes

 No

Depurar código Python no Visual Studio

Artigo • 18/04/2024

O Visual Studio proporciona uma experiência de depuração abrangente para Python. Neste artigo, você explora como anexar o depurador a processos em execução e avaliar expressões nas janelas **Observação** e **Imediata**. No depurador, você pode inspecionar variáveis locais, usar pontos de interrupção, instruções step in/out/over, **Definir Próxima Instrução** e muito mais.

Para obter informações de depuração específicas do cenário, consulte estes artigos:

- [Depuração remota do Linux](#)
- [Depuração de modo misto do Python/C++](#)
- [Símbolos para a depuração de modo misto](#)

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Código Python para usar com o depurador.

Depurar código com ou sem projeto

Se você quiser controlar o ambiente Python e argumentos, primeiro crie um projeto para seu código. Você pode criar um projeto com o modelo de projeto **Com base em um código existente do Python**. Para obter mais informações, consulte o [Crie um projeto a partir de arquivos de código Python existentes](#).

Porém, você não precisa de um arquivo de projeto ou solução no Visual Studio para depurar seu código Python. Para depurar código em um arquivo Python autônomo, abra o arquivo no Visual Studio e selecione **Depurar>Iniciar depuração**. O Visual Studio inicia o script com o ambiente padrão global e sem argumentos. Depois, você tem suporte de depuração completo para seu código. Para saber mais, veja [Ambientes do Python](#).

Explorar a depuração básica

O fluxo de trabalho básico de depuração envolve a definição de pontos de interrupção, a execução do código em etapas, a inspeção de valores e o tratamento de exceções.

Você pode iniciar uma sessão de depuração selecionando **Debug>Iniciar depuração** ou use o atalho de teclado **F5**. Para um projeto, essas ações iniciam o *arquivo de inicialização* com o ambiente ativo do projeto e quaisquer argumentos de linha de comando ou caminhos de pesquisa especificados para **Propriedades do Projeto**. Para configurar as propriedades, consulte [Definir opções de depuração de projeto](#).

Definir o arquivo de inicialização do projeto

O arquivo de inicialização de um projeto é mostrado em negrito no **Gerenciador de Soluções**. Você pode escolher qual arquivo usar como o arquivo de inicialização.

- Para especificar um arquivo de projeto como o arquivo de inicialização, clique com o botão direito do mouse no arquivo e selecione **Definir como item de inicialização**.

No Visual Studio 2017 versão 15.6 e posterior, você verá um alerta caso não tenha um conjunto de arquivos de inicialização especificado. Versões anteriores do Visual Studio podem abrir uma janela **Saída** com o interpretador Python em execução ou a janela **Saída** abre e fecha brevemente.

Especificar o ambiente ativo

Se você estiver usando um arquivo de projeto, o depurador sempre iniciará com o ambiente Python ativo para o projeto. Você pode mudar o ambiente ativo atual. Para obter mais informações, confira [Selecionar um ambiente Python para um projeto](#).

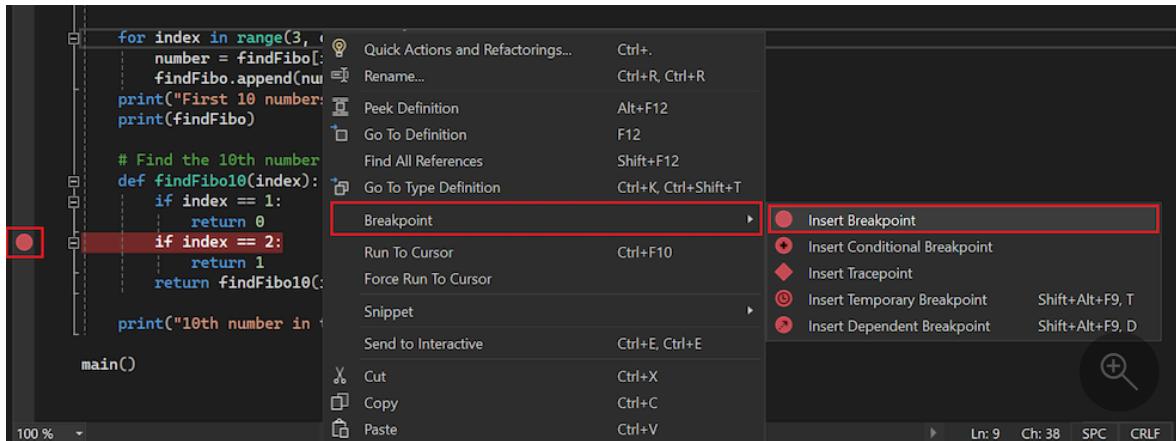
Se você estiver depurando um arquivo de código Python autônomo, o Visual Studio iniciará o script com o ambiente padrão global sem argumentos.

Definir Pontos de Interrupção

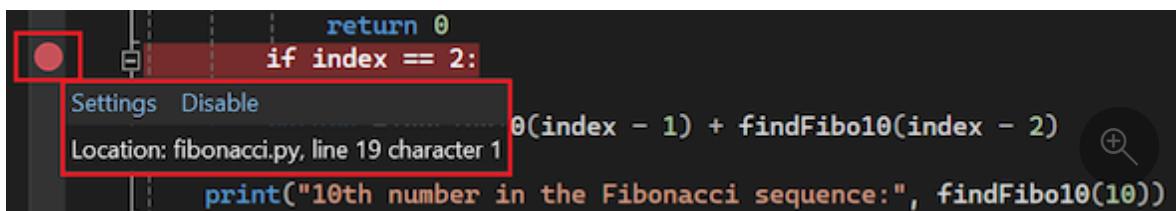
Os pontos de interrupção interrompem a execução de código em um ponto marcado, para que você possa inspecionar o estado do programa.

Alguns pontos de interrupção no Python podem ser surpreendentes para desenvolvedores que trabalharam com outras linguagens de programação. No Python, todo o arquivo é um código executável, para que o Python execute o arquivo quando ele é carregado para processar as definições de classe ou de função de nível superior. Se um Ponto de interrupção estiver definido, você poderá descobrir que o depurador está interrompendo uma declaração de classe parcialmente. Esse comportamento é correto, mesmo que às vezes seja surpreendente.

- Para definir um Ponto de interrupção, selecione na margem esquerda do editor de código ou clique com o botão direito em uma linha de código e selecione **Ponto de interrupção**>**Insert Breakpoint**. Um ponto vermelho é exibido em cada linha com Ponto de interrupção.



- Para remover um Ponto de interrupção, selecione o ponto vermelho ou clique com o botão direito na linha de código e selecione **Ponto de interrupção**>**Excluir Ponto de interrupção**. Você também pode desabilitar um Ponto de interrupção selecionando o ponto vermelho e selecionando **Ponto de interrupção**>**Desabilitar Ponto de interrupção**.

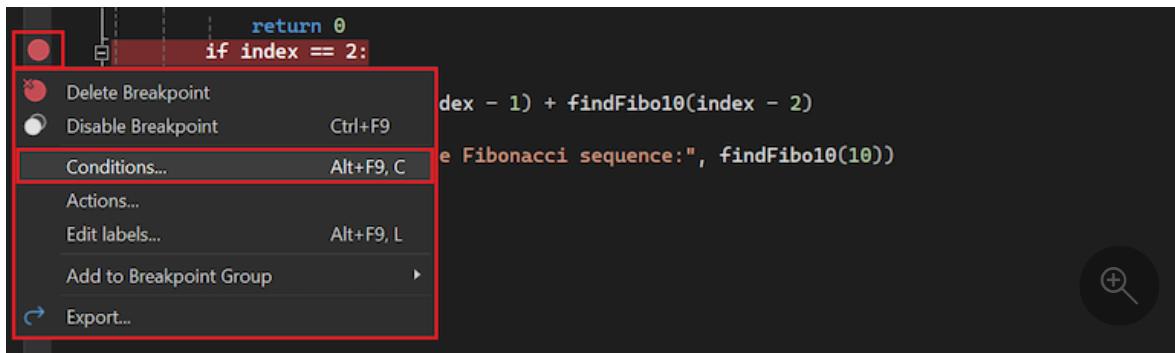


Definir condições e ações

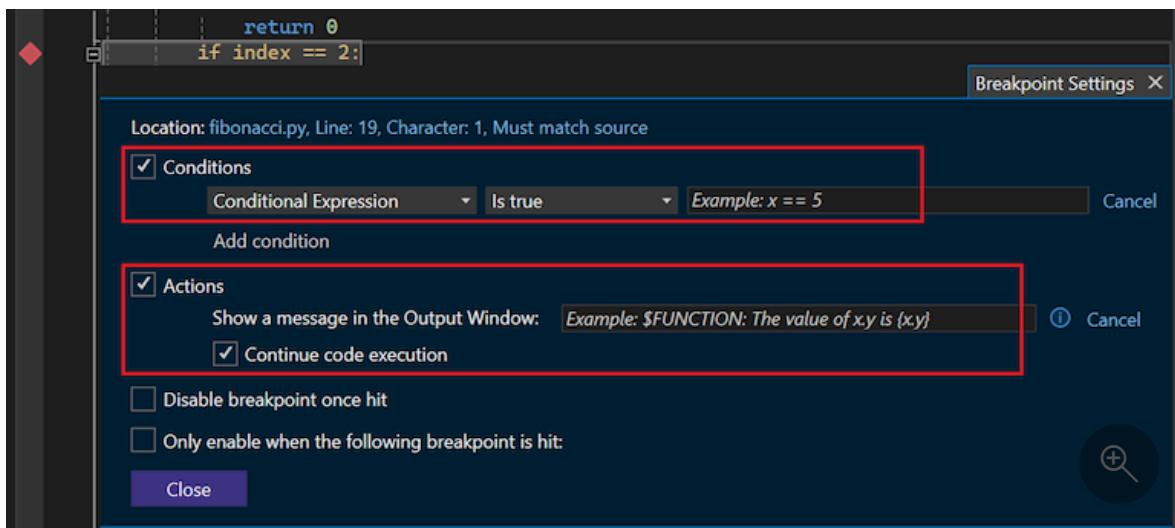
É possível personalizar as condições nas quais um ponto de interrupção é disparado, como a interrupção somente quando uma variável é configurada para um determinado valor ou intervalo de valores.

- Para definir condições, clique com o botão direito no ponto vermelho do ponto de interrupção e selecione **Condições**. A caixa de diálogo **Configurações do ponto de interrupção** é aberta.

Na caixa de diálogo, você pode adicionar várias condições e criar expressões condicionais com o código Python. Para obter detalhes completos sobre esse recurso no Visual Studio, confira [Condições de ponto de interrupção](#).



- Você também tem as opções para definir **Ações** para um Ponto de interrupção. Você pode criar uma mensagem para registrar na janela Saída e, opcionalmente, especificar para continuar a execução automaticamente.



Registrar uma mensagem cria o que um *tracepoint* que não adiciona código de log ao aplicativo diretamente.

Dependendo de como você configura as condições e ações para um ponto de interrupção, o ícone vermelho na margem esquerda muda para indicar suas configurações. Você pode ver a forma do ponto, um cronômetro ou um diamante.

Percorrer o código

Quando o Visual Studio interrompe a execução de código em um ponto de interrupção, há diversos comandos que você pode usar para percorrer seu código ou executar blocos de código antes de quebrar novamente. Os comandos estão disponíveis em alguns lugares no Visual Studio, incluindo a barra de ferramentas **Depurador**, o menu **Depurar**, o menu de contexto do botão direito do mouse no editor de código e pelos atalhos de teclado.

Depois, a tabela resume esses comandos e fornece o atalho de teclado:

| Comando | Atalho | Descrição |
|---------------------------|----------------|--|
| Continuar | F5 | Execute o código até chegar ao próximo ponto de interrupção. |
| Intervir | F11 | Execute a próxima instrução e parada. Se a próxima instrução for uma chamada a uma função, o depurador parará na primeira linha da função chamada. |
| Contornar | F10 | Execute a próxima instrução, incluindo fazer uma chamada a uma função (executando todo o código) e aplicar qualquer valor retornado. Este comando permite a depuração parcial permitindo ignorar facilmente as funções que não precisam ser depuradas. |
| Sair | Shift+F11 | Execute o código até o final da função atual e, depois, vá para a instrução de chamada. Esse comando é útil quando não é necessário depurar o restante da função atual. |
| Executar até o cursor | Ctrl+F10 | Execute o código até a localização do cursor no editor. Esse comando permite ignorar facilmente um segmento de código que não precisa ser depurado. |
| Definir Próxima Instrução | Ctrl+Shift+F10 | Altere o ponto de execução atual no código para a localização atual do cursor. Esse comando permite omitir a execução de um segmento de código, como nos casos em que você sabe que o código tem uma falha ou produz um efeito colateral indesejado. |
| Mostrar Próxima Instrução | Alt+Num+\ / | Volte à próxima instrução a ser executada no código. Esse comando ajuda a localizar o lugar no código onde o depurador está parado. |

Inspecionar e modificar valores

Ao interromper a execução do código no depurador, você poderá inspecionar e modificar os valores das variáveis. Use também a janela **Inspecção** para monitorar variáveis individuais e expressões personalizadas. Para obter mais informações, consulte [Inspecionar variáveis](#).

- Para exibir um valor usando o recurso **DataTips** durante a depuração, basta passar o mouse sobre qualquer variável no editor. Você pode selecionar o valor da variável para alterá-lo:

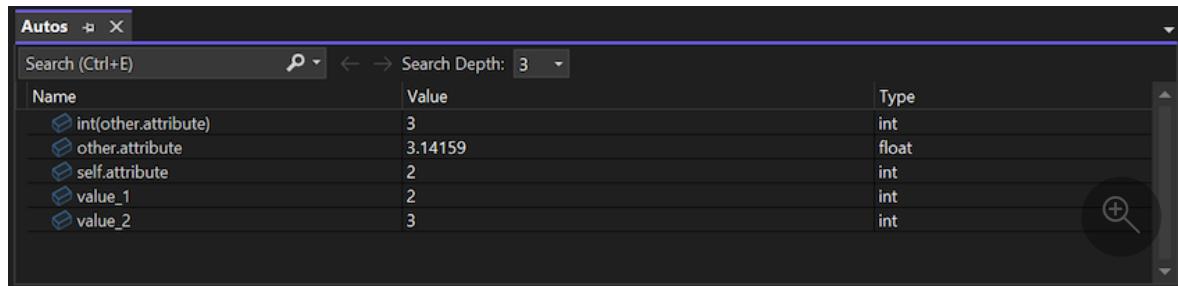
```

# Find the 10th number in the Fibonacci sequence
def findFibo10(index):
    if index == 1:
        return index
    if index == 2:
        return 1
    return findFibo10(index - 1) + findFibo10(index - 2)

```

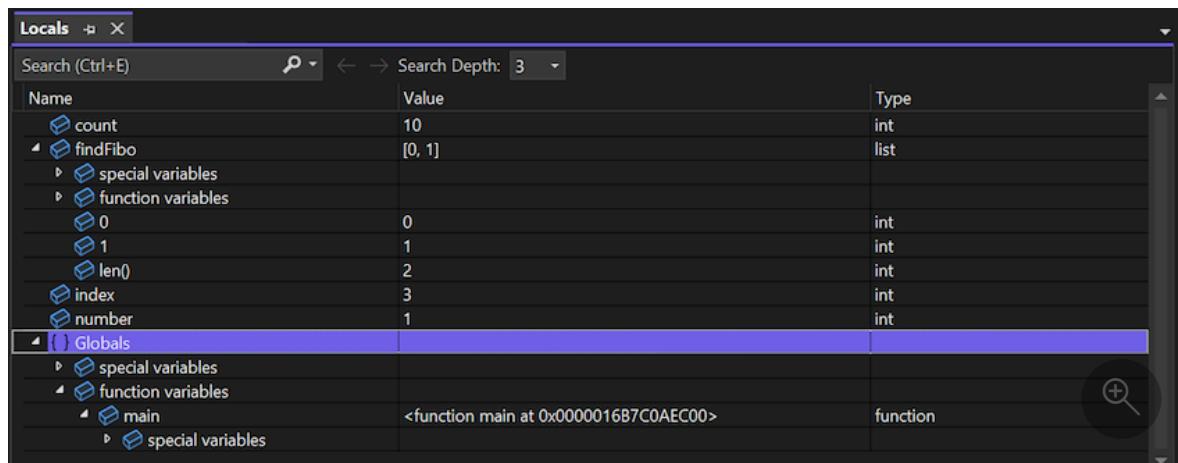
| Name | Type |
|----------------------|-------|
| int(other.attribute) | int |
| other.attribute | float |
| self.attribute | int |
| value_1 | int |
| value_2 | int |

- Para usar a janela **Automáticos**, selecione **Depurar>Windows>Automáticos**. Esta janela contém variáveis e expressões próximas da instrução atual. Clique duas vezes na coluna do valor ou selecione e pressione **F2** para editar o valor:



Para obter mais informações sobre como usar a janela **Automáticos**, consulte [Inspecionar variáveis nas janelas Automáticos e Locais](#).

- Para usar a janela **Locais**, selecione **Depurar>Windows>Locais**. Esta janela mostra todas as variáveis que estão no escopo atual, que podem ser editadas novamente:



Para obter mais informações sobre como usar a janela **Locais**, consulte [Inspecionar variáveis nas janelas Automáticos e Locais](#).

- Para usar as janelas **Assistir**, selecione **Depurar>Windows>Assistir>Assistir 1-4**. Essa opção permite que você insira expressões arbitrárias do Python e veja os resultados. As expressões são reavaliadas para cada etapa:

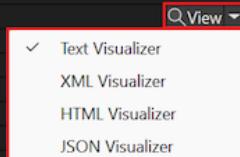
| Watch 1 | | |
|-----------------------|--------------|-------|
| Name | Value | Type |
| index+1 | x | |
| ↳ index | 3 | int |
| ↳ index-1 | 2 | int |
| ↳ index-2 | 1 | int |
| ↳ range(3, count + 1) | range(3, 11) | range |
| ↳ special variables | | |
| ↳ function variables | | |
| ↳ start | 3 | int |
| ↳ step | 1 | int |
| ↳ stop | 11 | int |
| Add item to watch | | |

Para obter mais informações sobre como usar a janela **Inspeção**, consulte [Definir uma inspeção em variáveis usando as janelas Inspeção e QuickWatch](#).

- Para inspecionar um valor de cadeia de caracteres, selecione **Exibir** (lupa) no lado direito da entrada **Valor**. Os tipos `str`, `unicode`, `bytes`, e `bytearray` estão todos disponíveis para inspeção.

O menu suspenso **Exibir** mostra quatro opções de visualização: Texto, HTML, XML ou JSON.

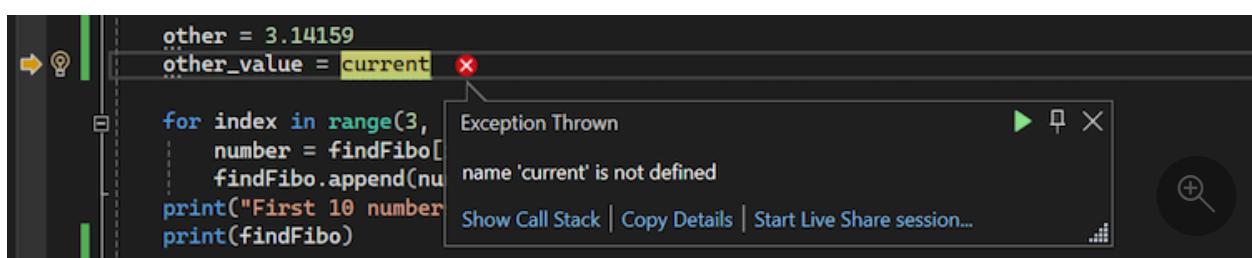
| Autos | | |
|----------------------|-------------------------|-----------|
| Name | Value | Type |
| value_2 | 3 | int |
| ↳ utf16Fibo | <bytearray, len() = 68> | bytearray |
| ↳ special variables | | |
| ↳ function variables | | |
| ↳ utf8Fibo | <bytearray, len() = 33> | bytearray |
| ↳ special variables | | |
| ↳ function variables | | |



Depois de selecionar uma visualização, uma caixa de diálogo pop-up exibe o valor da cadeia de caracteres sem aspas de acordo com o tipo selecionado. Você pode exibir a cadeia de caracteres com a quebra automática e rolagem, realce de sintaxe e exibições em árvore. Essas visualizações podem ajudar a depurar problemas de cadeias de caracteres longas e complexas.

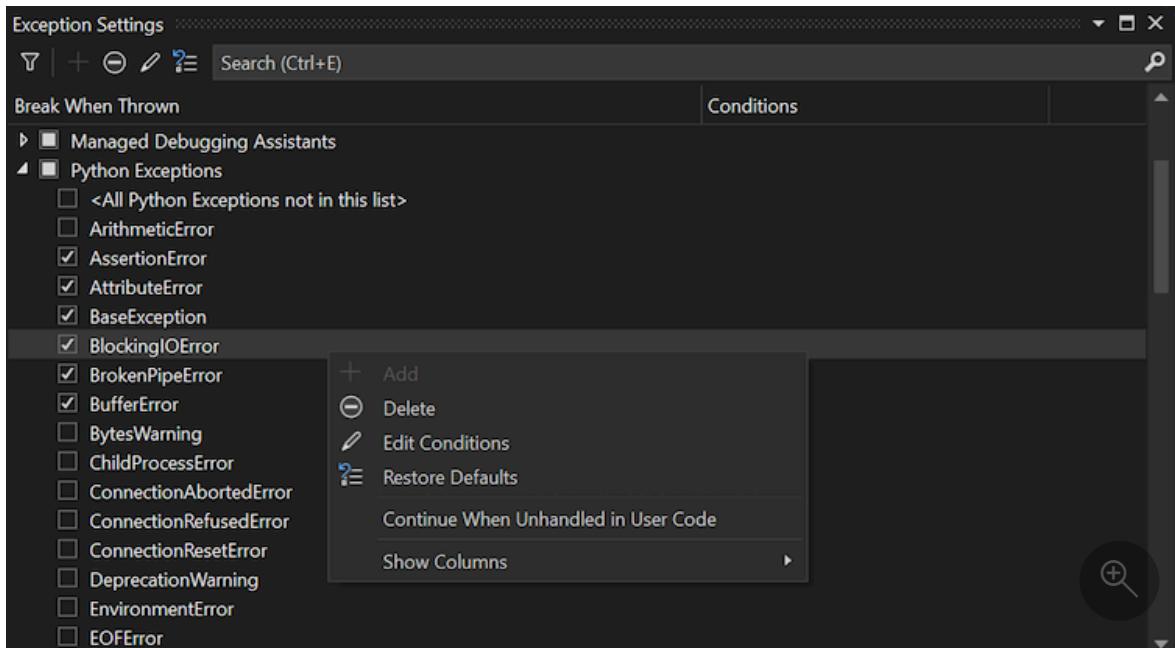
Exibir exceções

Se ocorrer um erro no programa durante a depuração, mas você não tiver um manipulador de exceção para ele, o depurador interromperá no ponto da exceção:



Quando ocorrer um erro, você pode inspecionar o estado atual do programa, incluindo a pilha de chamadas. Porém, se você percorrer o código, o processo de depuração continuará lançando a exceção até que ela seja tratada ou o programa seja encerrado.

- Para ver uma exibição expandida de exceções, selecione **Depurar>Windows>Configurações de exceção**.



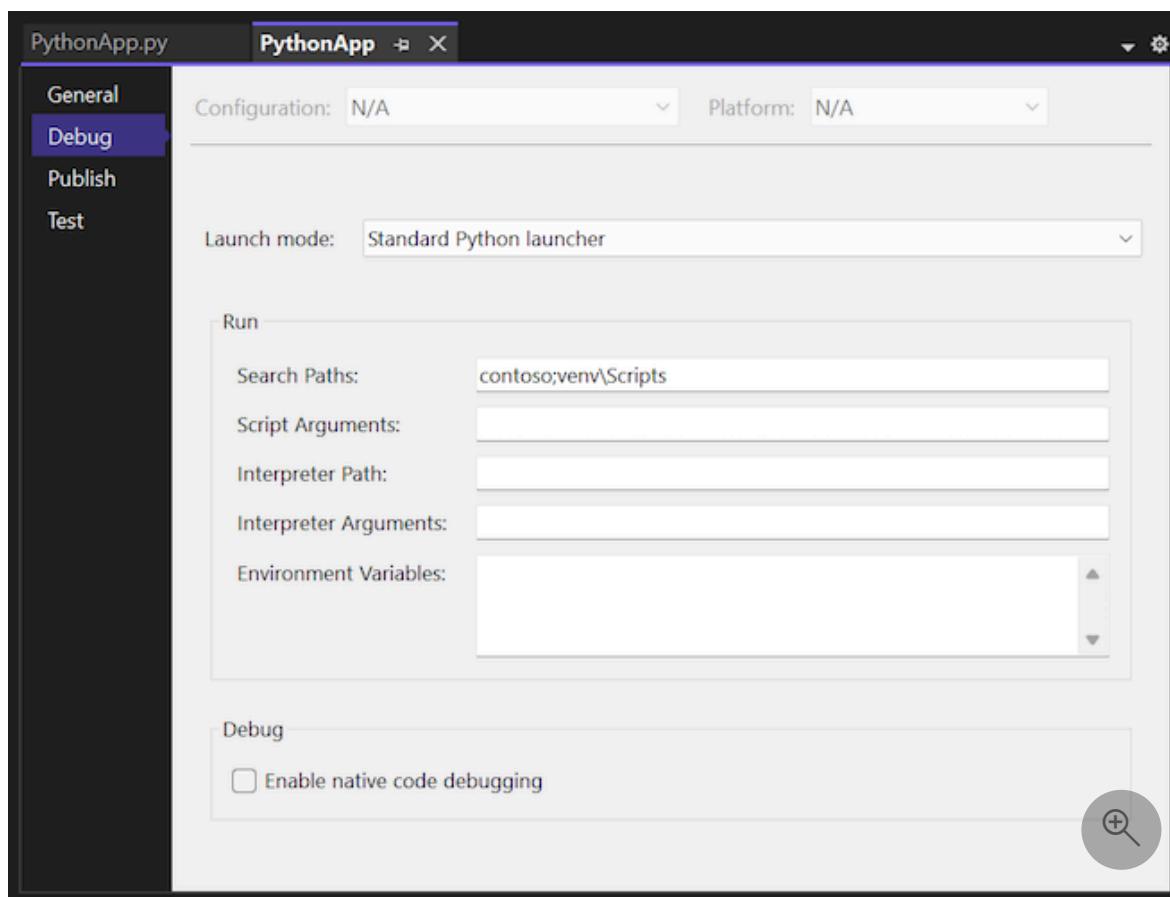
Na janela **Configurações de exceções**, a caixa de seleção ao lado de uma exceção controla se o depurador *always* quebra quando essa exceção é gerada.

- Para quebrar com mais frequência para uma exceção específica, marque a caixa de seleção ao lado da exceção na janela **Configurações de exceção**.
- Por padrão, a maioria das exceções interromperá quando não for possível encontrar um manipulador de exceção no código-fonte. Para alterar esse comportamento, clique com o botão direito do mouse em qualquer exceção e modifique a opção **Continuar Quando Não For Tratada no Código do Usuário**. Para quebrar com menos frequência para a exceção, desmarque a opção.
- Para configurar uma exceção que não aparece na janela **Configurações de Exceção**, selecione **Adicionar** (símbolo de adição). Digite um nome para a exceção a ser observada. O nome deve corresponder ao nome completo da exceção.

Configurar opções de depuração de projeto

Por padrão, o depurador inicia o programa com o inicializador padrão do Python, sem argumentos de linha de comando e sem nenhum outro caminho ou condição especial. Você pode configurar as opções de inicialização para um projeto Python ao definir as propriedades de depuração.

- Para acessar as propriedades de depuração de um projeto, clique com o botão direito do mouse no projeto Python no **Gerenciador de soluções**, selecione **Propriedades**, e selecione a guia **Depurador**.



As seções seguintes descrevem as propriedades específicas.

Definir comportamento de inicialização

A tabela a seguir lista os valores possíveis para a propriedade **Modo de inicialização**. Use esta propriedade para definir o comportamento de inicialização do depurador.

[\[+\] Expandir a tabela](#)

| Valor | Descrição |
|--------------------------------|---|
| Inicializador padrão do Python | Use o código de depuração escrito no Python portátil que é compatível com o CPython, IronPython e variantes como o Stackless Python. Essa opção fornece a melhor experiência de depuração de código puro do Python. Quando você o anexa a um processo <code>python.exe</code> em execução, o Launcher especificado nessa propriedade é usado. Esse iniciador também fornece a depuração de modo misto para o CPython, que permite a execução em etapas direta entre o código do C/C++ e o código do Python. |
| Inicializador da Web | Inicie o navegador padrão na inicialização e habilita a depuração de modelos. Para mais informações, consulte a seção Depuração de modelos da Web . |

| Valor | Descrição |
|------------------------------------|--|
| Inicializador da Web do Django | Implemente um comportamento idêntico à propriedade Web launcher para um ambiente Django. Use esta opção somente para fins de compatibilidade com versões anteriores. |
| Inicializador do IronPython (.NET) | Use o depurador do .NET, que funciona somente com o IronPython, mas que permite a execução em etapas entre qualquer projeto de linguagem .NET, incluindo C# e Visual Basic. Esse inicializador é usado se você se anexar a um processo em execução do .NET que hospeda o IronPython. |

Definir comportamento de execução

A tabela a seguir descreve as propriedades que você pode definir para configurar o comportamento de execução do depurador.

[Expandir a tabela](#)

| Propriedade | Descrição |
|-----------------------------|---|
| Caminhos de Pesquisa | Especifique os caminhos de pesquisa de arquivo e pasta que o Visual Studio usa para o seu projeto. Esses valores correspondem os itens mostrados no nó Caminhos de Pesquisa do projeto no Gerenciador de Soluções . Embora você possa especificar caminhos de pesquisa nessa caixa de diálogo, pode ser mais fácil usar o Gerenciador de Soluções , onde você pode navegar nas pastas e converter automaticamente os caminhos em formato relativo. |
| Argumentos do script | Defina os argumentos a serem adicionados ao comando que o Visual Studio usa para iniciar seu script e aparecem após o nome do arquivo do seu script. O primeiro item listado no valor está disponível para o script como <code>sys.argv[1]</code> , o segundo como <code>sys.argv[2]</code> e assim por diante. |
| Argumentos do interpretador | Liste os argumentos para adicionar à linha de comando do inicializador antes do nome do script. Os argumentos comuns aqui são <code>-w ...</code> para controlar avisos, <code>-o</code> para otimizar ligeiramente o programa e <code>-u</code> para usar o E/S não armazenado em buffer. Provavelmente, os usuários do IronPython usarão esse campo para passar opções <code>-X</code> , como <code>-X:Frames</code> ou <code>-X:MTA</code> . |
| Caminho do Interpretador | Identifique um caminho de intérprete para substituir o caminho associado ao ambiente presente. O valor poderá ser útil para iniciar o script com um interpretador não padrão. |
| Variáveis de Ambiente | Use essa propriedade para adicionar entradas do formato <code><NAME>=\<VALUE></code> . O Visual Studio aplica esse valor de propriedade por último, sobre qualquer variável de ambiente global existente, e depois <code>PYTHONPATH</code> é definido de acordo com a configuração Caminhos de pesquisa . Como resultado, essa configuração pode ser usada para substituir manualmente qualquer uma destas outras variáveis. |

Trabalhe com janelas interativas

Há duas janelas **interativas** que podem ser usadas durante uma sessão de depuração: a janela padrão do Visual Studio e a janela **Imediata** e a janela **Interativa de Depuração do Python**.

Abrir a janela Imediata

Você pode usar a janela padrão **Imediata** do Visual Studio para avaliar rapidamente expressões Python e inspecionar ou atribuir variáveis em seu programa em execução. Para obter mais informações, consulte [Janela Imediata](#).

- Para abrir a janela **Immediate**, selecione **Depurar>Windows>Imediata**. Você também pode usar o atalho de teclado **Ctrl+Alt+I**.

Abra a Janela Interativa de Depuração

A **Janela Interativa de Depuração** oferece um ambiente rico com a experiência completa [REPL Interativo](#) interativo disponível durante a depuração, incluindo a gravação e execução de código. Essa janela se conecta automaticamente a qualquer processo iniciado no depurador usando o inicializador Padrão do Python, incluindo os processos anexados por meio de **Depurar>Anexar ao Processo**. Porém, esta janela não está disponível ao usar a depuração C/C++ de modo misto.

- Para usar a janela **Interativa de Depuração**, selecione **Depurar>Windows>Interativa de Depuração do Python (Shift+Alt+I)**.

```

fibonacci.py  X
    print("First 10 numbers in the Fibonacci sequence:")
    count = 1

    # Find the 10th number in the Fibonacci sequence
    def findFibo10(index, count):
        # Count function access
        count = count + 1

        if index == 1:
            return 0
        if index == 2:
            return 1
        return findFibo10(index - 1, count) + findFibo10(index - 2, count)

100 %  No issues found  Ln: 27 Ch: 1 SPC CRLF
Autos Locals Watch 1 Debug Interactive 3  X
>>> count
2
>>> count > index
False
>>> index
10
>>> |

```

A janela **Interativa de Depuração** suporta meta-comandos especiais além do [comandos REPL padrão](#), conforme descrito na tabela a seguir:

[] [Expandir a tabela](#)

| Comando | Descrição |
|---|---|
| <code>\$continue</code> , <code>\$cont</code> , <code>\$c</code> | Inicie a execução do programa da instrução atual. |
| <code>\$down</code> , <code>\$d</code> | Move o quadro atual um nível para baixo no rastreamento de pilha. |
| <code>\$frame</code> | Exiba a ID de quadro atual. |
| <code>\$frame</code> | Mude a ID de quadro atual para a ID de quadro especificada. - Requer um argumento de <i><ID de quadro></i> . |
| <code>\$load</code> | Carregue comandos do arquivo e os executa até a conclusão. |
| <code>\$proc</code> | Exiba a ID de processo atual. |
| <code>\$proc</code> | Mude a ID de processo atual para a ID de processo especificada. - Requer um argumento de <i><ID de processo></i> . |
| <code>\$procs</code> | Liste os processos que estão sendo depurados no momento. |
| <code>\$stepin</code> , <code>\$step</code> , <code>\$s</code> | Intervenha na próxima chamada de função, se possível. |
| <code>\$stepout</code> , <code>\$return</code> , <code>\$r</code> | Encaminhe-se para fora da função atual. |
| <code>\$stepover</code> , <code>\$until</code> , <code>\$unt</code> | Depure parcialmente a próxima chamada de função. |

| Comando | Descrição |
|--------------------|---|
| \$thread | Exiba a ID de thread atual. |
| \$thread | Mude a ID de thread atual para a ID de thread especificada. - Requer um argumento de <ID de thread>. |
| \$threads | Liste os threads que estão sendo depurados no momento. |
| \$up, \$u | Move o quadro atual um nível para cima no rastreamento de pilha. |
| \$where, \$w, \$bt | Liste os quadros do thread atual. |

As janelas padrão do depurador, como **Processos**, **Threads** e **Pilha de Chamadas**, não são sincronizadas com a janela **Interativa de Depuração**. Se você alterar o processo, thread ou quadro ativo no **Debug Interactive** não afeta as outras janelas do depurador. Da mesma forma, a alteração do processo, do thread ou do quadro ativo nas outras janelas do depurador não afeta a janela **Interativa de Depuração**.

Usar o depurador herdado

Dependendo da configuração do ambiente, talvez seja preciso usar o depurador herdado:

- O Visual Studio 2017 versão 15.7 e anterior com Python 2.6, 3.1 a 3.4 ou IronPython
- O Visual Studio 2019 versão 16.5 e posterior com Python 2.6, 3.1 a 3.4 ou IronPython
- ptvsd 3.x e versões anteriores 4.x

O depurador herdado é o padrão no Visual Studio 2017 versão 15.7 e anterior.

- Para usar o depurador herdado, selecione **Ferramentas**>**Opções**, expanda as opções **Python**>**Depurando** e selecione a opção **Usar depurador legado**.

Suporte a versões mais antigas do Visual Studio ou Python

O Visual Studio 2017 versão 15.8 e posteriores usam um depurador com base no ptvsd versão 4.1 e superior. O Visual Studio 2019 versão 16.5 e posteriores usam um depurador baseado no debugpy. Essas duas versões do depurador são compatíveis com o Python 2.7 ou Python 3.5 e superiores.

Se você estiver executando uma dessas versões do Visual Studio, mas estiver usando Python 2.6, 3.1 a 3.4 ou IronPython, o Visual Studio mostrará o erro **O depurador não dá suporte a este ambiente do Python**:

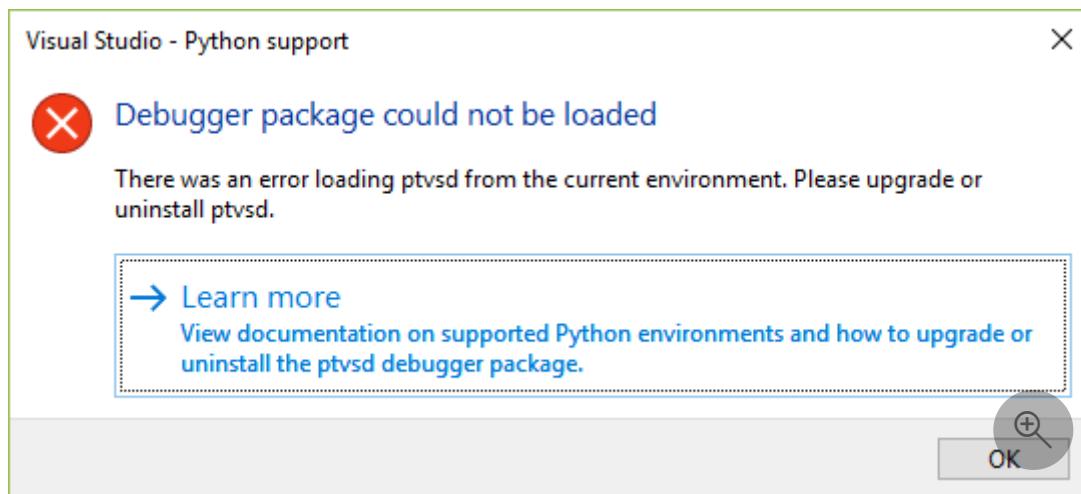


Quando o Visual Studio informa esse erro de ambiente, você deve usar o depurador herdado.

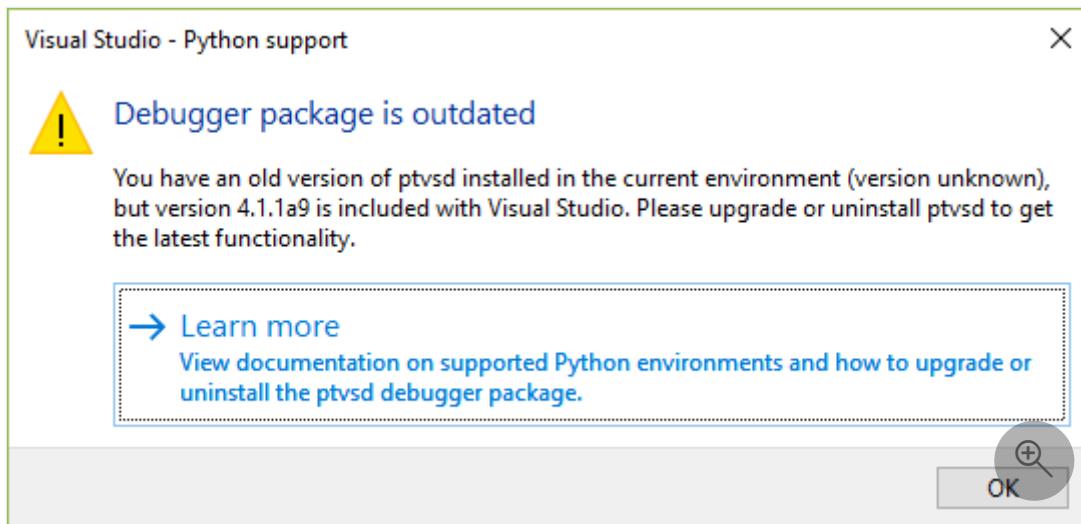
Supporte a versões ptvsd mais antigas

Se você tiver usando uma versão mais antiga do ptvsd no ambiente atual (como uma versão 4.0.x anterior ou uma versão 3.x necessária para a depuração remota), o Visual Studio poderá mostrar um erro ou aviso.

Se seu ambiente usa ptvsd 3.x, Visual Studio mostra o erro **Não foi possível carregar o pacote do depurador**:



O aviso O pacote do depurador está desatualizado será exibido se você estiver usando uma versão 4.x anterior do ptvsd:



Quando o Visual Studio informa esses erros de ambiente, você deve usar o depurador herdado.

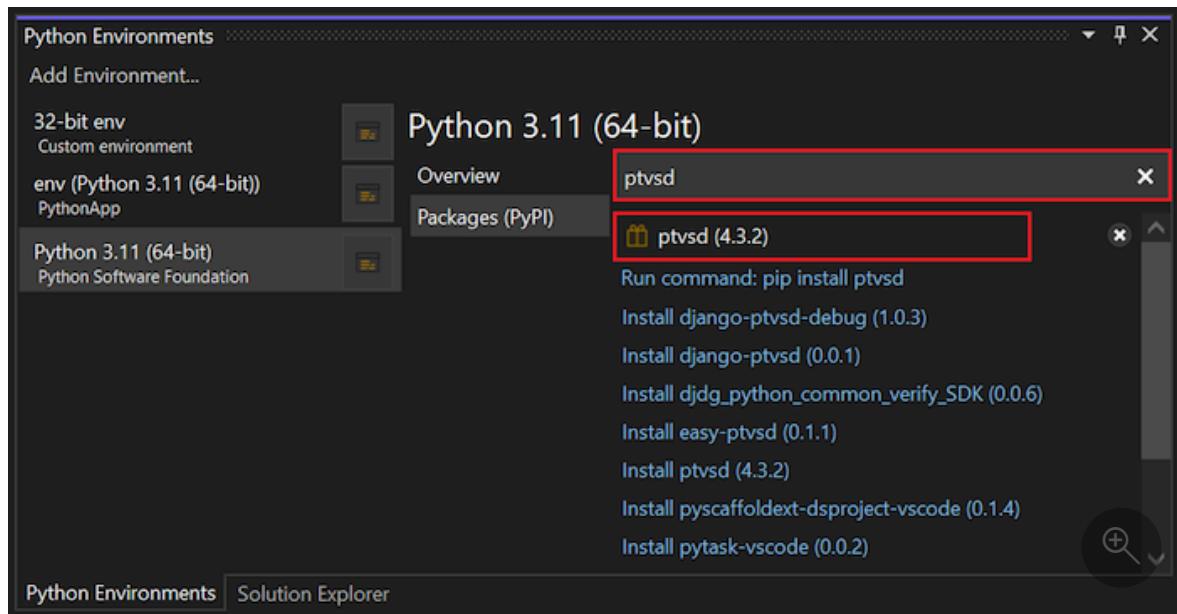
ⓘ Importante

Embora você possa optar por ignorar o aviso em algumas versões do ptvsd, o Visual Studio pode não funcionar corretamente.

Gerenciar a instalação do ptvsd

Siga estes passos para gerenciar sua instalação ptvsd:

1. Na janela **Ambientes do Python**, acesse a guia **Pacotes**.
2. Insira *ptvsd* na caixa de pesquisa e examine a versão do ptvsd instalada:



3. Se a versão for inferior à 4.1.1a9 (a versão empacotada com o Visual Studio), selecione o X à direita do pacote para desinstalar a versão mais antiga. Assim, o Visual Studio passará a usar a versão empacotada. Você também pode desinstalar do PowerShell usando o comando `pip uninstall ptvsd`.
4. Como alternativa, você pode atualizar o pacote ptvsd para a última versão seguindo as instruções na seção [Solução de problemas](#).

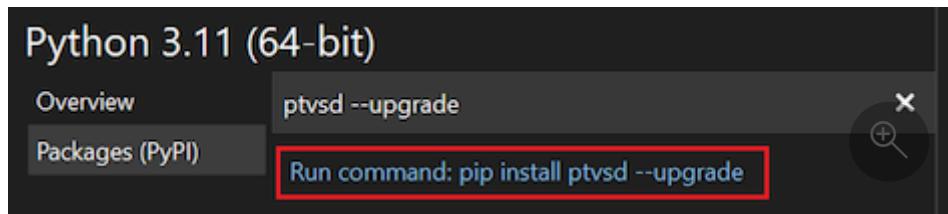
Solução de problemas de cenários de depuração

Os cenários a seguir descrevem outras opções de solução de problemas para a configuração de depuração.

Atualizar o ptvsd para Visual Studio 2019

Se você tiver problemas com o depurador no Visual Studio 2019 versão 16.4 e anterior, atualize primeiro sua versão do depurador da seguinte maneira:

1. Na janela **Ambientes do Python**, acesse a guia **Pacotes**.
2. Insira `ptvsd --upgrade` na caixa de pesquisa, em seguida, selecione **Executar o comando: pip install ptvsd --upgrade**. (Você também pode usar o mesmo comando no PowerShell).



Se o problema persistir, registre o problema no [Repositório GitHub do PTVS](#).

➊ Observação

No Visual Studio 2019 versão 16.5 e posteriores, a depuração faz parte da carga de trabalho do Python no Visual Studio e é atualizada junto com o Visual Studio.

Habilitar registro em log do depurador

Ao investigar o problema do depurador, a Microsoft pode solicitar que você habilite e colete logs de depurador para ajudar no diagnóstico.

As seguintes etapas habilitam a depuração na sessão atual do Visual Studio:

1. Abra uma janela de comando no Visual Studio selecionando **Visualizar>Outros Windows>Janela de Comando**.
2. Insira o seguinte comando:

```
Console  
DebugAdapterHost.Logging /On /OutputWindow
```

3. Inicie a depuração e veja todas as etapas necessárias para reproduzir o problema. Durante esse tempo, os logs de depuração aparecem na janela **Saída em Log de Host do Adaptador de Depuração**. Depois, você poderá copiar os logs dessa janela e colá-los em um problema do GitHub, em um email etc.

```
Output  
Show output from: Debug  
list index out of range  
Stack trace:  
> File "C:\source\repos\PythonApplication1\Standalone\fibonacci.py", line 14, in main  
>     number = findFibo[index - 2] + findFibo[index - 3]  
>     ~~~~~~  
> File "C:\source\repos\PythonApplication1\Standalone\fibonacci.py", line 43, in <module>  
>     main()  
> IndexError: list index out of range  
Loaded '__main__'  
Loaded 'runpy'
```

Command Window | Output | Debug Interactive 3 | Watch 1 | Autos | Locals

4. Se o Visual Studio parar de responder ou se não for possível acessar a janela Saída, reinicie o Visual Studio, abra uma janela de comando e digite o seguinte comando:

Console

```
DebugAdapterHost.Logging /On
```

5. Inicie a depuração e reproduza o problema novamente. Os logs do depurador ficam em `%temp%\DebugAdapterHostLog.txt`.

Conteúdo relacionado

- [Depurando no Visual Studio](#)
- [Depuração de modo misto do Python/C++](#)
- [Símbolos para a depuração de modo misto](#)

Comentários

Esta página foi útil?

 Yes

 No

Depurar remotamente o código Python no Linux no Visual Studio

Artigo • 18/04/2024

Neste artigo, você explora como configurar a instalação do Visual Studio para dar suporte à depuração de código Python em computadores Linux remotos. Este passo a passo é baseado no Visual Studio 2019 versão 16.6.

O Visual Studio pode iniciar e depurar aplicativos Python local e remotamente em um computador Windows. O Visual Studio também dá suporte à depuração também pode depurar remotamente em um dispositivo ou sistema operacional diferente ou em uma implementação Python que não seja o CPython usando a [biblioteca debugpy](#).

O Visual Studio 2019 versão 16.4 e anterior usa a [biblioteca ptvsd](#). No Visual Studio 2019 versão 16.5 e posterior, a biblioteca debugpy substitui a ptvsd. Quando você usa a debugpy, o código do Python que está sendo depurado hospeda o servidor de depuração ao qual o Visual Studio pode se anexar. Essa hospedagem exige uma pequena modificação no seu código para importar e habilitar o servidor. Talvez também seja preciso ajustar as configurações de rede ou firewall no computador remoto para permitir as conexões TCP.

Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Um computador remoto que execute o Python em um sistema operacional como o Mac OSX ou Linux.
- A porta 5678 (entrada) abre no firewall desse computador remoto, que é o padrão para a depuração remota.

Configurar um computador Linux

Crie com facilidade uma [máquina virtual do Linux no Azure](#) e [acesse-a usando a Área de Trabalho Remota](#) no Windows. O Ubuntu para a máquina virtual é conveniente porque o Python é instalado por padrão. Se você tiver outra configuração, consulte [Instalar interpretadores do Python](#) para outros locais de download do Python.

Configurar o firewall

A porta de entrada 5678 deve estar aberta no firewall do computador remoto para dar suporte à depuração remota.

Para ver em detalhes como criar uma regra de firewall para uma máquina virtual do Azure, consulte estes artigos:

- [Filtrar tráfego de rede com um grupo de segurança de rede usando o Portal do Azure](#)
- [Rotear tráfego com uma tabela de rotas utilizando o Portal do Azure](#)
- [Implantar e configurar o Firewall do Azure usando o portal do Azure](#)

Preparar o script para depuração

Siga estes passos para preparar um script para depurar seu código Python no Linux.

1. No computador remoto, crie um arquivo do Python chamado *guessing-game.py* com este código:

```
Python

import random

guesses_made = 0
name = input('Hello! What is your name?\n')
number = random.randint(1, 20)
print('Well, {0}, I am thinking of a number between 1 and
20.'.format(name))

while guesses_made < 6:
    guess = int(input('Take a guess: '))
    guesses_made += 1
    if guess < number:
        print('Your guess is too low.')
    if guess > number:
        print('Your guess is too high.')
    if guess == number:
        break
if guess == number:
    print('Good job, {0}! You guessed my number in {1}
guesses!'.format(name, guesses_made))
else:
    print('Nope. The number I was thinking of was {0}'.format(number))
```

2. Instale o pacote `debugpy` no ambiente usando o comando `pip3 install debugpy`.

ⓘ Observação

É recomendável gravar a versão do debugpy que está instalada caso você precise dela para solução de problemas. A [listagem debugpy](#) também mostra as versões disponíveis.

3. Habilite a depuração remota adicionando o seguinte código na parte superior do arquivo *guessing-game.py* antes de outro código. (Embora esse não seja um requisito estrito, é impossível depurar os threads em segundo plano gerados antes que a função `listen` seja chamada.)

Python

```
import debugpy  
debugpy.listen(('0.0.0.0', 5678))
```

4. Salve o arquivo e execute o programa:

Python

```
python3 guessing-game.py
```

A chamada para a função `listen` é executada em segundo plano e aguarda as conexões de entrada enquanto você interage com o programa de outra maneira. Se quiser, você pode chamar a função `wait_for_client` depois de chamar a função `listen` para bloquear o programa até que o depurador seja anexado.

💡 Dica

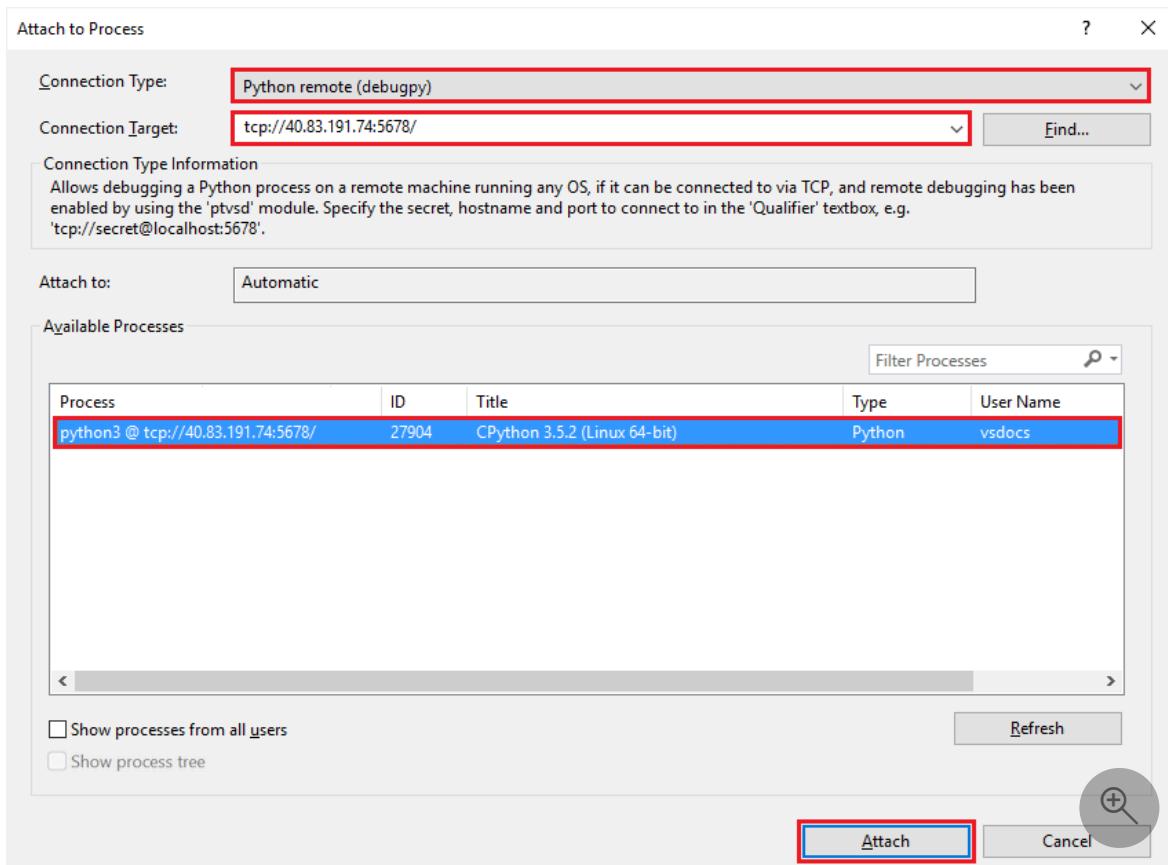
Além das funções `listen` e `wait_for_client`, o debugpy também fornece uma função `breakpoint`. Essa função serve como ponto de interrupção programático se o depurador estiver conectado. Outra função, `is_client_connected`, retorna `True` se o depurador for anexado. Você não precisa verificar esse resultado antes de chamar outras funções `debugpy`.

Anexar remotamente por meio das Ferramentas Python

Os passos a seguir mostram como definir um ponto de interrupção para interromper o processo remoto.

1. Crie uma cópia do arquivo remoto no computador local e abra-a no Visual Studio.
Não importa a localização do arquivo, mas o nome deve corresponder ao nome do script no computador remoto.
2. (Opcional) Para ter o IntelliSense para debugpy no computador local, instale o pacote de debugpy em seu ambiente de Python.
3. Selecione **Depurar>Anexar ao Processo**.
4. Na caixa de diálogo **Anexar ao processo**, defina **Tipo de conexão** como **Python remoto (debugpy)**.
5. No campo **Destino da conexão**, insira o comando `tcp://<ip_address>:5678`.
 - O `tcp://` especifica o tipo de conexão como Protocolo TCP (protocolo de controle de transmissão).
 - `<ip_address>` é o endereço IP do computador remoto; ele pode ser um endereço explícito ou um nome como `myvm.cloudapp.net`.
 - `:5678` é o número da porta de depuração remota.

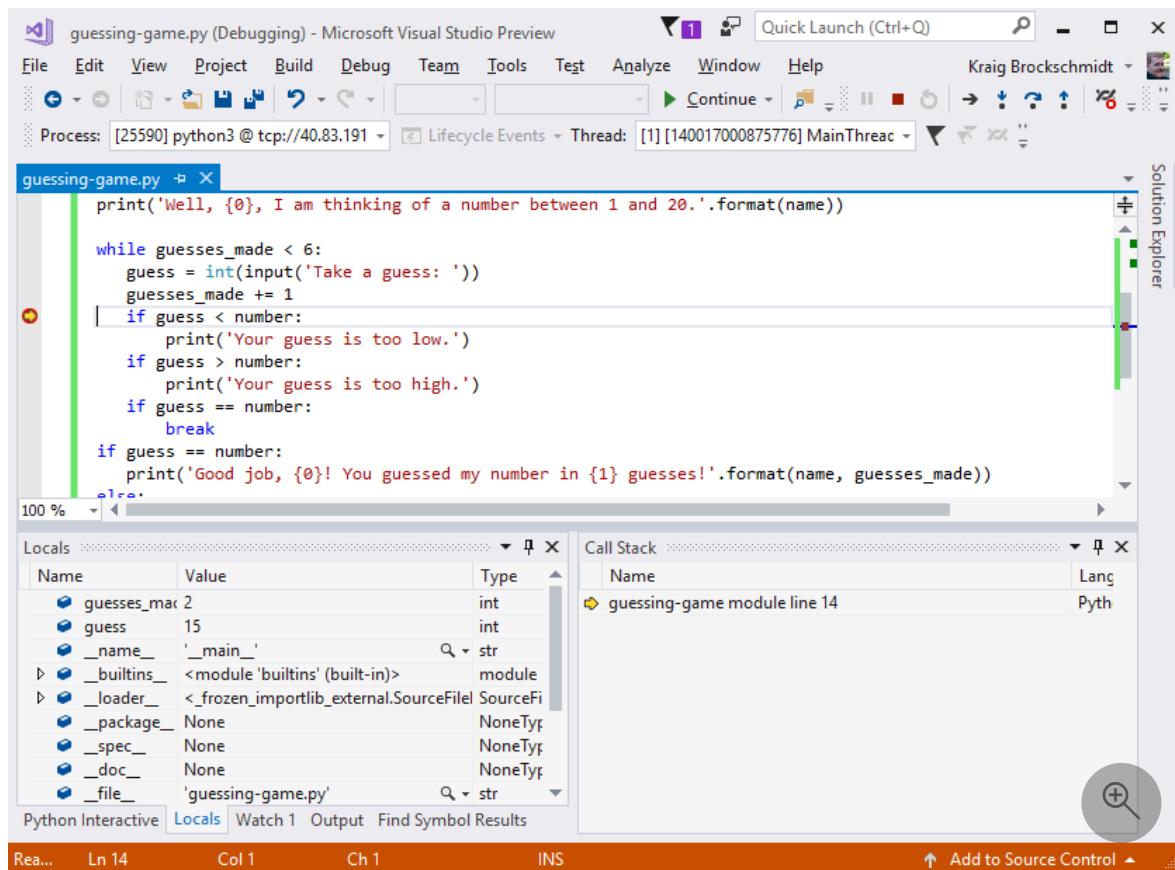
6. Selecione **Enter** para popular a lista de processos do debugpy disponíveis nesse computador:



Se outro programa for iniciado no computador remoto após a lista ser populada, selecione o botão **Atualizar**.

7. Selecione o processo a ser depurado e selecione **Anexar** ou clique duas vezes no processo.
8. Então, o Visual Studio alternará para o modo de depuração enquanto o script continuará a ser executado no computador remoto, fornecendo todos os recursos normais de **depuração**.

Você pode definir um ponto de interrupção na linha `if guess < number:` e, em seguida, mude para o computador remoto e insira outra tentativa. O Visual Studio do seu computador local parará no ponto de interrupção, mostrará variáveis locais e assim por diante:



9. Quando você parar de depurar, o Visual Studio irá se desanexar do programa. O programa continua sendo executado no computador remoto. O debugpy também continua a escuta para anexar depuradores, assim, é possível anexá-los novamente ao processo a qualquer momento.

Solucionar problemas de conexão

Analise os seguintes pontos para ajudar a solucionar problemas com a conexão.

- Certifique-se de selecionar **Python remota (debugpy)** para **Tipo de Conexão**
- Confirme que o segredo no **Destino de Conexão** corresponde exatamente ao segredo no código remoto.
- Confirme o endereço IP no **Destino de Conexão** corresponde ao do computador remoto.
- Verifique se a porta de depuração remota do computador remoto está aberta e se o destino da conexão inclui o sufixo da porta, como `:5678`.

Para usar outra porta, especifique o número da porta na chamada para a função `listen`, como em `debugpy.listen((host, port))`. Nesse caso, certifique-se de abrir a porta específica no firewall.

- Confirme se a versão de depuração instalada no computador remoto (conforme retornado pelo comando `pip3 list`) corresponde à versão do Visual Studio Python Tools (PTVS).

A tabela a seguir lista os pares de versões válidos. Conforme necessário, atualize no computador remoto a versão do debugpy.

 Expandir a tabela

| Visual Studio | Ferramentas do Python | debugpy |
|---------------|-----------------------|---------|
| 2019 16.6 | 1.0.0b5 | 1.0.0b5 |
| 2019 16.5 | 1.0.0b1 | 1.0.0b1 |

Observação

O Visual Studio 2019 versão 16.0-16.4 utilizou ptvsd, não debugpy. O processo neste passo a passo para essas versões é semelhante, mas os nomes das funções são diferentes. O Visual Studio 2019 versão 16.5 usa depuração, mas os nomes de função eram iguais aos do ptvsd. Em vez de `listen`, você usaria `enable_attach`. Em vez de `wait_for_client`, você usaria `wait_for_attach`. Em vez de `breakpoint`, você usaria `break_into_debugger`.

Usar ptvsd 3.x para depuração herdada

O depurador herdado do ptvsd 3.x é o padrão no Visual Studio 2017 versão 15.7 e anterior.

Dependendo da configuração do Visual Studio, talvez seja preciso usar o ptvsd 3.x para depuração remota:

- O Visual Studio 2017 versão 15.7 e anterior com Python 2.6, 3.1 a 3.4 ou IronPython
- O Visual Studio 2019 versão 16.5 e posterior com Python 2.6, 3.1 a 3.4 ou IronPython
- Versões 4.x anteriores

Se sua configuração implementar um cenário de versão mais antiga, o Visual Studio mostrará o erro, **O depurador não dá suporte a este ambiente do Python.**

Configurar depuração remota

Para se preparar para a depuração remota com o ptvsd 3.x, siga estes passos:

1. Configure seu segredo, que é usado para restringir acesso ao script em execução.

No ptvsd 3.x, a função `enable_attach` requer que você passe um "segredo" como o primeiro argumento.

- Ao anexar o depurador remoto, insira o segredo com o comando
`enable_attach(secret=<secret>)`.

Embora você possa permitir que qualquer pessoa se conecte usando o comando `enable_attach(secret=None)`, essa opção não é recomendada.

2. Crie a sua URL de destino de conexão no formato

`tcp://<secret>@<ip_address>:5678`.

- `tcp://` especifica o tipo de conexão como TCP.
- `<secret>` é a cadeia de caracteres passada com a função `enable_attach` no código Python.
- `<ip_address>` é o endereço IP do computador remoto; ele pode ser um endereço explícito ou um nome como `myvm.cloudapp.net`.
- `:5678` é o número da porta de depuração remota.

Conexão segura com protocolo TCPS

Por padrão, a conexão com o servidor de depuração remota ptvsd 3.x é protegida somente pelo segredo e todos os dados são passados em texto sem formatação. Para uma conexão mais segura, o ptvsd 3.x dá suporte a SSL usando a forma segura do protocolo TCP ou *TCPS*.

Use estes passos para configurar o ptvsd 3.x para funcionar com o protocolo TCPS:

1. No computador remoto, use o comando `openssl` para gerar arquivos separados para a chave e o certificado autoassinado:

Console

```
openssl req -new -x509 -days 365 -nodes -out cert.cer -keyout cert.key
```

- No prompt, `openssl` digite o nome do host ou o endereço IP que você usa para se conectar ao **Nome comum**.

Para obter mais informações, consulte [Certificados autoassinados](#) na documentação do módulo Python `ssl`. Observe que o comando descrito na documentação do Python gera somente um único arquivo combinado.

2. No código, modifique a chamada para a função `enable_attach` para incluir os argumentos `certfile` e `keyfile` usando os nomes de arquivos como os valores. Esses argumentos têm o mesmo significado que para a função Python padrão `ssl.wrap_socket`.

Python

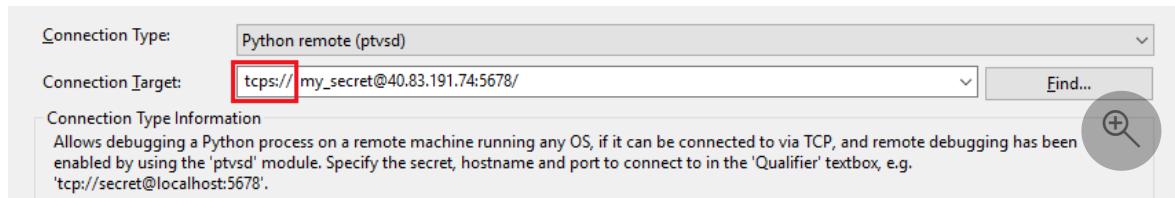
```
ptvsd.enable_attach(secret='my_secret', certfile='cert.cer',  
keyfile='cert.key')
```

Você também pode fazer a mesma modificação no arquivo de código no computador local. Como esse código não é executado, ele não é estritamente necessário.

3. Reinicie o programa de Python no computador remoto para que fique pronto para depuração.
4. Proteja o canal adicionando o certificado à AC Raiz Confiável no computador Windows com o Visual Studio:
 - a. Copie o arquivo de certificado do computador remoto para o computador local.

- b. Abra o Painel de Controle e acesse Ferramentas do Windows>Gerenciar certificados de computador.
- c. Na caixa de diálogo certlm [Certificados - computador local], expanda o nó Autoridades de Certificação Confiáveis, clique com o botão direito em Certificados e selecione Todas as tarefas>Importar.
- d. Procure e selecione o arquivo .cer copiado do computador remoto.
- e. Continue pelos prompts da caixa de diálogo para concluir o processo de importação.
5. Repita o processo de anexação no Visual Studio, como descrito anteriormente em [Anexar remotamente por meio das Ferramentas Python](#).

Para este caso, defina `tcps://` como o protocolo para o **Destino da conexão** (ou **Qualificador**).



Resolver os problemas de conexão

Ao tentar a conexão, o Visual Studio pode encontrar problemas. Analise os cenários a seguir e aplique a ação apropriada, conforme necessário.

- O Visual Studio avisa sobre possíveis problemas de certificado ao se conectar via SSL.

Ação: você pode ignorar a mensagem e continuar.

✖ Cuidado

Lembre-se de que, embora o canal ainda esteja criptografado contra espionagem, ele pode estar aberto a ataques de man-in-the-middle.

- O Visual Studio exibe o aviso de **certificado remoto não confiável**.

Problema: o certificado não foi adicionado corretamente à AC de Raiz Confiável.

Ação: verifique novamente os passos para [adicionar o certificado à AC de Raiz Confiável no computador Windows](#) e tente a conexão novamente.

Visual Studio - Python support



Could not establish secure connection to
tcp://my_secret@40.83.191.74:5678/?&opt=WaitOnAbnormalExit,
WaitOnNormalExit, RedirectOutput, RichExceptions because of the
following SSL issues:
- remote certificate is not trusted

Connect anyway?

Yes

No



- O Visual Studio mostra o aviso **O nome do certificado remoto não corresponde ao nome do host.**

Problema: o nome de host ou endereço IP apropriado não está especificado para o **Nome comum** do certificado.

Ação: verifique novamente os passos em [Proteger a conexão com TCPS](#).

Certifique-se de usar o **Nome comum correto** ao criar o certificado e tente a conexão novamente.

Visual Studio - Python support



Could not establish secure connection to
tcp://my_secret@40.83.191.74:5678/?&opt=WaitOnAbnormalExit,
WaitOnNormalExit, RedirectOutput, RichExceptions because of the
following SSL issues:
- remote certificate name does not match hostname

Connect anyway?

Yes

No



Conteúdo relacionado

- [Depuração remota](#)

Comentários

Esta página foi útil?

Yes

No

Criar uma extensão C++ para o Python no Visual Studio

Artigo • 18/04/2024

Neste artigo, veja como criar um módulo de extensão C++ para CPython que calcula uma tangente hiperbólica e faz uma chamada a ela do código Python. A rotina é implementada primeiro em Python para demonstrar o ganho de desempenho relativo da implementação da mesma rotina em C++.

Os módulos escritos em C++ (ou C) são normalmente usados para estender os recursos de um interpretador do Python. Há três tipos principais de módulos de extensão:

- **Módulos de acelerador:** permitem um desempenho acelerado. Como Python é uma linguagem interpretada, você pode escrever módulos de acelerador no C++ para obter um desempenho mais alto.
- **Módulos de wrapper:** expõem interfaces C/C++ existentes para o código Python ou expõem uma API mais "Pythonic" fácil de usar do Python.
- **Módulos de acesso de baixo nível do sistema:** criam módulos de acesso ao sistema para acessar os recursos de baixo nível do runtime `cPython`, do sistema operacional ou do hardware subjacente.

Este artigo também demonstra duas maneiras de disponibilizar um módulo de extensão C++ para Python:

- Utilize as extensões `cPython` padrão, conforme descrito na [Documentação Python](#).
- Utilize [PyBind11](#), recomendamos para C++ 11 devido à sua simplicidade. Para garantir a compatibilidade, confirme que você esteja trabalhando com uma das versões mais recentes do Python.

O exemplo completo deste passo a passo está no GitHub em [python-samples-vs-cpp-extension](#).

Pré-requisitos

- O Visual Studio 2017 ou posterior com a carga de trabalho de desenvolvimento Python instalada. A carga de trabalho inclui as ferramentas de desenvolvimento nativo do Python, que adicionam a carga de trabalho do C++ e os conjuntos de ferramentas necessários para extensões nativas.

Installation details

- Visual Studio core editor
- ▼ Python development
 - Included
 - ✓ Python language support
 - Optional
 - GitHub Copilot
 - Python 3 64-bit (3.6.0)
 - Python native development tools
 - Python web support
 - Live Share
 - Azure Cloud Services core tools



Para obter mais informações sobre as opções de instalação, consulte [Instalar o suporte do Python para Visual Studio](#).

ⓘ Observação

Quando você instala a da carga de trabalho **Aplicativos de ciência de dados e análise**, o Python e a opção **Ferramentas nativas de desenvolvimento em Python** são instaladas por padrão.

- Se você instalar o Python separadamente, lembre-se de selecionar **Baixar símbolos de depuração** em **Opções Avançadas** no instalador Python. Essa opção é necessária para que você use a depuração de modo misto entre o código Python e o código nativo.

Criar o aplicativo do Python

Siga estes passos para criar o aplicativo Python.

1. Crie um novo projeto Python no Visual Studio escolhendo **Arquivo > Novo > Projeto**.
2. Na caixa de diálogo **Criar um novo projeto**, pesquise por *python*. Selecione o modelo **Aplicativo do Python** e **Avançar**.
3. Sira um **Nome do Projeto** e **Localização**, e selecione **Criar**.

O Visual Studio cria o projeto. O projeto é aberto no **Gerenciador de Soluções** e o arquivo de projeto (*.py*) é aberto no editor de códigos.

4. o arquivo `.py`, cole o código a seguir. Para experimentar alguns [recursos de edição do Python](#), tente inserir o código manualmente.

Esse código calcula uma tangente hiperbólica sem usar a biblioteca de matemática e é isso que você vai acelerar com extensões nativas Python.

Dica

Escreva seu código em Python puro antes de reescrevê-lo em C++. Dessa forma, você pode verificar com mais facilidade se o código nativo Python está correto.

Python

```
from random import random
from time import perf_counter

# Change the value of COUNT according to the speed of your computer.
# The value should enable the benchmark to complete in approximately 2
# seconds.
COUNT = 500000
DATA = [(random() - 0.5) * 3 for _ in range(COUNT)]

e = 2.7182818284590452353602874713527

def sinh(x):
    return (1 - (e ** (-2 * x))) / (2 * (e ** -x))

def cosh(x):
    return (1 + (e ** (-2 * x))) / (2 * (e ** -x))

def tanh(x):
    tanh_x = sinh(x) / cosh(x)
    return tanh_x

def test(fn, name):
    start = perf_counter()
    result = fn(DATA)
    duration = perf_counter() - start
    print('{} took {:.3f} seconds\n\n'.format(name, duration))

    for d in result:
        assert -1 <= d <= 1, "incorrect values"

if __name__ == "__main__":
    print('Running benchmarks with COUNT = {}'.format(COUNT))

    test(lambda d: [tanh(x) for x in d], '[tanh(x) for x in d] (Python
implementation)')
```

5. Execute o programa selecionando **Depurar>Iniciar sem Depuração** ou selecionando **Ctrl+F5**.

Uma janela de comando é aberta e mostra a saída do programa.

6. Na saída, veja a quantidade de tempo relatada para o processo de benchmark.

Para este passo a passo, o processo de benchmark deve levar cerca de 2 segundos.

7. Conforme necessário, ajuste o valor da variável `COUNT` no código para permitir que o benchmark seja concluído em cerca de 2 segundos no computador.

8. Execute o programa novamente e veja se o valor modificado `COUNT` produz o benchmark em cerca de 2 segundos.

Dica

Ao executar parâmetros de comparação, sempre use a opção **Depuração>Iniciar sem Depuração**. Este método ajuda a evitar a sobrecarga que você incorre ao executar o código no depurador do Visual Studio.

Criar os projetos principais de C++

Siga estes passos para criar dois projetos C++ idênticos, *superfastcode* e *superfastcode2*. Posteriormente, você usará uma abordagem diferente em cada projeto para expor o código C++ para Python.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó da solução e selecione **Adicionar>Novo projeto**.

Uma solução do Visual Studio pode conter os projetos Python e C++, que é uma das vantagens de usar o Visual Studio para desenvolvimento Python.

2. Na caixa de diálogo **Adicionar um novo projeto**, defina o filtro **Idioma** como **C++**, e insira *vazio* na caixa **Pesquisar**.

3. a lista de resultados do modelo de projeto, selecione **Projeto vazio**, e selecione **Avançar**.

4. Na caixa de diálogo **Configurar seu novo projeto**, insira o **Nome do projeto**:

- Para o primeiro projeto, insira o nome *superfastcode*.
- Para o segundo projeto, insira o nome *superfastcode2*.

5. Selecione Criar.

Lembre-se de repetir esses passos e criar dois projetos.

Dica

Uma abordagem alternativa fica disponível quando você tem as ferramentas de desenvolvimento nativo Python instaladas no Visual Studio. Você pode começar com o modelo **Módulo de extensão do Python**, que pré-conclui vários passos descritos neste artigo.

Para o passo a passo deste artigo, começar com um projeto vazio ajuda a demonstrar como construir o módulo de extensão passo a passo. Depois de entender o processo, você pode usar o modelo alternativo para poupar tempo ao escrever suas próprias extensões.

Adicione o arquivo C++ ao projeto

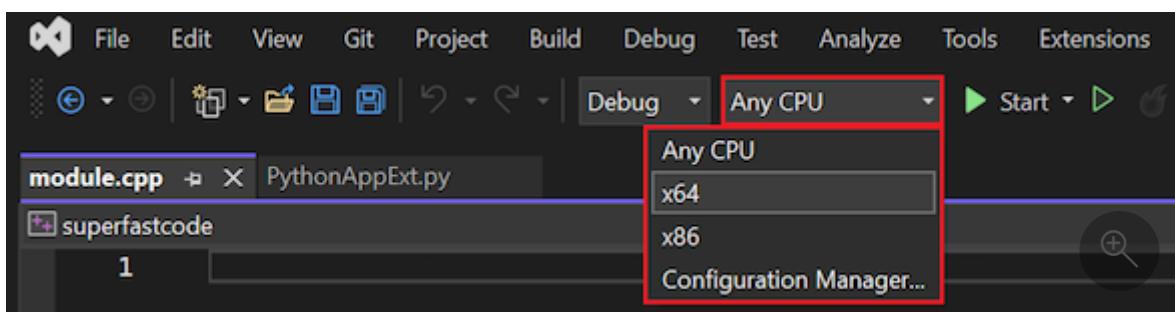
Depois, adicione um arquivo C++ a cada projeto.

1. Em **Gerenciador de Soluções**, expanda o projeto, clique com o botão direito do mouse no nó **Arquivos de origem** e selecione **Adicionar>Novo item**.
2. Na lista de modelos de arquivo, selecione o **Arquivo C++ (.cpp)**.
3. Insira o **Nome** para o arquivo como *module.cpp*, e selecione **Adicionar**.

Importante

Certifique-se de que o nome do arquivo inclua a extensão *.cpp*. O Visual Studio busca um arquivo com a extensão *.cpp* para habilitar a exibição das páginas de propriedades do projeto C++.

4. Na barra de ferramentas, expanda o menu suspenso **Configuração** e selecione o tipo de configuração de destino:



- Para um runtime do Python de 64 bits, ative a configuração **x64**.
- Para um runtime do Python de 32 bits, ative a configuração do **Win32**.

Lembre-se de repetir esses passos para os dois projetos.

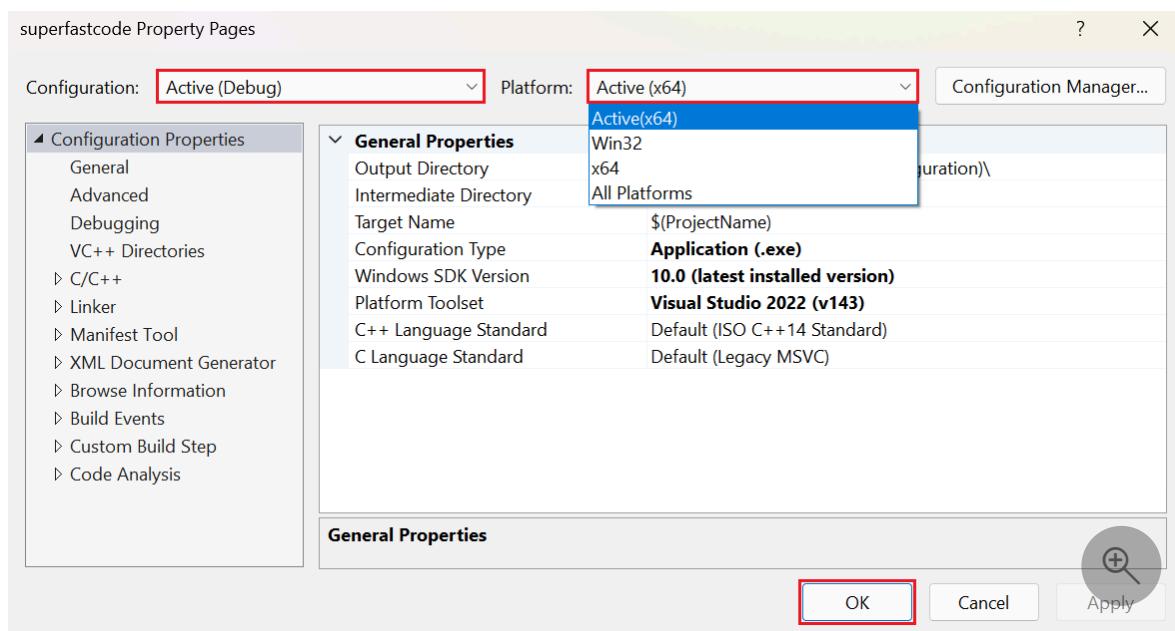
Configurar as propriedades do projeto

Antes de adicionar código aos novos arquivos C++, configure as propriedades de cada projeto de módulo C++ e teste as configurações para garantir que tudo funcione bem.

Você precisa definir as propriedades do projeto para as configurações de compilação de *depuração* e *versão* de cada módulo.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto do módulo (*superfastcode* ou *superfastcode2*) e selecione **Propriedades**.
2. Configure as propriedades para a compilação de *depuração* do módulo e, depois, configure as mesmas propriedades para a compilação de *versão*:

Na parte superior da caixa de diálogo **Páginas de propriedades do projeto**, configure as seguintes opções de configuração de arquivo:



- a. Para a **Configuração**, selecione **Depurar** ou **Versão**. (Você poderá ver essas opções com o prefixo **Ativo**.)
- b. Para a **Plataforma**, selecione **Ativo (x64)** ou **Ativo (Win32)**, dependendo da seleção na etapa anterior.

Observação

Ao criar seus próprios projetos, é recomendável definir as configurações de *depuração* e *versão* separadamente, de acordo com seus requisitos de cenário específicos. Neste exercício, você define as configurações para usar uma compilação da versão do CPython. Essa configuração desabilita alguns recursos de depuração do runtime do C++, incluindo asserções. O uso de binários de depuração do CPython (`python_d.exe`) requer configurações diferentes.

c. Defina outras propriedades do projeto conforme descrito na tabela a seguir.

Para mudar o valor de uma propriedade, insira um valor no campo de propriedade. Para alguns campos, você pode selecionar o valor atual para expandir o menu suspenso de opções ou abrir a caixa de diálogo para ajudar a definir o valor.

Depois de atualizar os valores em uma guia, selecione **Aplicar** antes de alternar para outra guia. Essa ação ajuda a garantir que suas alterações permaneçam.

 Expandir a tabela

| Guia e seção | Propriedade | Valor |
|---|-----------------------------------|---|
| Propriedades de Configuração > Geral | Nome de destino | Especifique o nome do módulo ao qual você deseja se referir do Python nas instruções <code>from...import</code> , como <code>superfastcode</code> . Você usa esse mesmo nome no código C++ ao definir o módulo para Python. Para usar o nome do projeto como o nome do módulo, deixe o valor padrão de <code>\$<ProjectName></code> . Para <code>python_d.exe</code> , adicione <code>_d</code> ao final do nome. |
| | Tipo de Configuração | Biblioteca Dinâmica (.dll) |
| Propriedades de Configuração > Avançada | Extensão do arquivo de destino | .pyd (Módulo de Extensão do Python) |
| C/C++ > Geral | Diretórios de Inclusão Adicionais | Adicione a pasta <i>include</i> do Python conforme apropriado para sua instalação (por exemplo, <code>c:\Python36\include</code>). |

| Guia e seção | Propriedade | Valor |
|---------------------------|-------------------------------------|---|
| C/C++ > Pré-processador | Definições do Pré-processador | Se estiver presente, mude o valor <code>_DEBUG</code> para <code>NDEBUG</code> para corresponder à versão de não depuração do CPython. Quando você usar <code>python_d.exe</code> , deixe esse valor inalterado. |
| C/C++ > Geração de Código | Biblioteca em Runtime | DLL de vários threads (/MD) para corresponder à versão (não depuração) do CPython. Quando você usar <code>python_d.exe</code> , deixe esse valor como DLL de depuração multi-threaded (/MDd). |
| | Verificações básicas de runtime | Padrão |
| Vinculador > Geral | Diretórios de Biblioteca Adicionais | Adicione a pasta <code>libs</code> do Python que contém arquivos <code>.lib</code> conforme apropriado para sua instalação (por exemplo, <code>c:\Python36\libs</code>). Lembre-se de apontar para a pasta <code>libs</code> que contém arquivos <code>.lib</code> e não para a pasta <code>Lib</code> que contém arquivos <code>.py</code> . |

ⓘ Importante

Se a guia **C/C++** não for exibida como opção para as propriedades do projeto, o projeto não conterá arquivo de código que o Visual Studio identifique como arquivos de origem C/C++. Essa condição poderá ocorrer se você criar um arquivo de origem sem uma extensão de arquivo `.c` ou `.cpp`.

Se você inseriu acidentalmente `module.coo` em vez de `module.cpp` quando você criou o arquivo C++, o Visual Studio cria o arquivo, mas não define o tipo para *Compilador C/C+*. Esse tipo de arquivo é necessário para ativar a presença da guia Propriedades C/C++ na caixa de diálogo das propriedades do projeto. A identificação incorreta permanece mesmo que você renomeie o arquivo de código com uma extensão de arquivo `.cpp`.

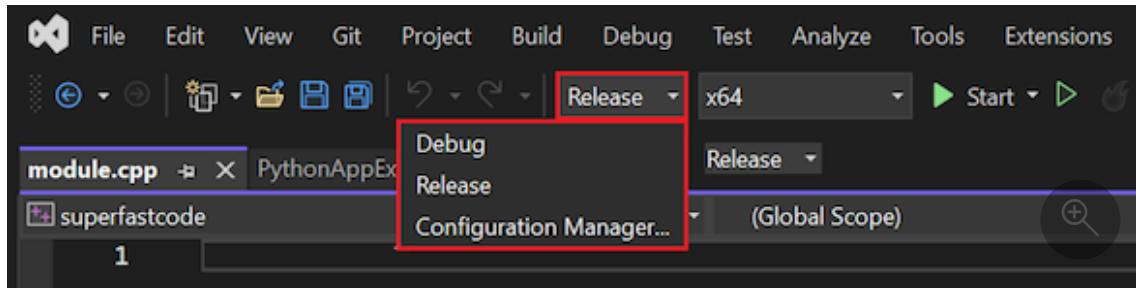
Para definir o tipo de arquivo de código corretamente, em **Gerenciador de Soluções**, clique com o botão direito do mouse no arquivo de código e selecione **Propriedades**. Para o **Tipo de item**, selecione **Compilador C/C++**.

d. Depois de atualizar todas as propriedades, selecione **OK**.

Repita estes passos para a outra configuração de build.

3. Testar a configuração atual. Repita as etapas a seguir para as compilações *depuração* e *versão* dos dois projetos C++.

a. Na barra de ferramentas do Visual Studio, defina a configuração de **Compilação** como **Depurar** ou **Versão**:



b. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto C++ e selecione **Compilar**.

Os arquivos *.pyd* estão localizados na pasta *solução* em *Depurar* e *Versão*, e não na própria pasta do projeto do C++.

Adicionar código e testar configuração

Agora você está pronto para adicionar código aos seus arquivos C++ e testar a compilação da *versão*.

1. Para o projeto C++ *superfastcode*, abra o arquivo *module.cpp* no editor de código.

2. No arquivo *module.cpp*, cole estes código:

```
C++  
  
#include <Windows.h>  
#include <cmath>  
  
const double e = 2.7182818284590452353602874713527;  
  
double sinh_impl(double x) {  
    return (1 - pow(e, (-2 * x))) / (2 * pow(e, -x));  
}  
  
double cosh_impl(double x) {  
    return (1 + pow(e, (-2 * x))) / (2 * pow(e, -x));  
}  
  
double tanh_impl(double x) {
```

```
    return sinh_impl(x) / cosh_impl(x);
}
```

3. Salve suas alterações.

4. Crie a configuração de *versão* para o projeto C++ para confirmar que o seu código esteja correto.

Repita as etapas para adicionar código ao arquivo C++ para o projeto *superfastcode2* e teste a compilação *versão*.

Converter os projetos de C++ em uma extensão de Python

Para transformar a DLL C++ em uma extensão para Python, modifique primeiro os métodos exportados para interagir com os tipos Python. Depois, adicione uma função para exportar o módulo, juntamente com as definições dos métodos do módulo.

As seções a seguir mostram como criar as extensões usando as extensões CPython e PyBind11. O projeto *superfasctcode* usa as extensões CPython e o projeto *superfasctcode2* implementa PyBind11.

Usar extensões CPython

Para mais informações sobre o código apresentado nesta seção, consulte o [Manual de Referência da API do Python/C](#) e, principalmente, a página [Objetos de Módulo](#). Ao ler o conteúdo de referência, lembre-se de selecionar sua versão do Python na lista suspensa no canto superior direito.

1. Para o projeto C++ *superfastcode*, abra o arquivo *module.cpp* no editor de código.
2. Adicione uma instrução na parte superior do arquivo *module.cpp* para incluir o arquivo de cabeçalho *Python.h*:

```
C++

#include <Python.h>
```

3. Substitua código do método `tanh_impl` para aceitar e retornar tipos Python (ou seja, um `PyObject*`):

```
C++
```

```
PyObject* tanh_impl(PyObject* /* unused module reference */, PyObject* o) {
    double x = PyFloat_AsDouble(o);
    double tanh_x = sinh_impl(x) / cosh_impl(x);
    return PyFloat_FromDouble(tanh_x);
}
```

4. Ao final do arquivo, adicione uma estrutura para definir como apresentar a função C++ `tanh_impl` ao Python:

```
C++

static PyMethodDef superfastcode_methods[] = {
    // The first property is the name exposed to Python, fast_tanh
    // The second is the C++ function with the implementation
    // METH_O means it takes a single PyObject argument
    { "fast_tanh", (PyCFunction)tanh_impl, METH_O, nullptr },

    // Terminate the array with an object containing nulls
    { nullptr, nullptr, 0, nullptr }
};
```

5. Adicione outra estrutura para definir como se referir ao módulo no seu código Python, especificamente quando você usa a instrução `from...import`.

O nome que está sendo importado nesse código deve corresponder ao valor nas propriedades do projeto em **Propriedades de Configuração>Geral>Nome de Destino**.

No exemplo a seguir, o nome `"superfastcode"` significa que você pode usar a declaração `from superfastcode import fast_tanh` no Python, pois `fast_tanh` é definido dentro de `superfastcode_methods`. Os nomes de arquivo internos do projeto C++, como `module.cpp`, não são importantes.

```
C++

static PyModuleDef superfastcode_module = {
    PyModuleDef_HEAD_INIT,
    "superfastcode", // Module name to use with
    Python import statements
    "Provides some functions, but faster", // Module description
    0,
    superfastcode_methods // Structure that defines
    the methods of the module
};
```

6. Adicione um método que o Python chama ao carregar o módulo. O nome do método deve ser `PyInit_<module-name>`, em que `<module-name>` corresponde exatamente à propriedade **Propriedades de Configuração**>**Geral**>**Nome de destino** do projeto C++. Ou seja, o nome do método corresponde ao nome do arquivo `.pyd` criado pelo projeto.

C++

```
PyMODINIT_FUNC PyInit_superfastcode() {  
    return PyModule_Create(&superfastcode_module);  
}
```

7. Compile o projeto C++ e verifique seu código. Se encontrar erros, confira a seção "["Solucionar erros de compilação"](#)".

Usar PyBind11

Se concluir as etapas na seção anterior para o projeto *superfastcode*, você poderá notar que o exercício requer o código clichê para criar as estruturas de módulo para extensões CPython C++. Neste exercício, você descobrirá que o PyBind11 simplifica o processo de codificação. Você usa macros em um arquivo de cabeçalho C++ para chegar ao mesmo resultado, mas com muito menos código. Porém, é preciso executar outros passos para garantir que o Visual Studio possa localizar as bibliotecas PyBind11 e incluir arquivos. Para obter mais informações sobre o código nesta seção, consulte [Noções básicas do PyBind11 ↴](#).

Instalar o PyBind11

O primeiro passo é instalar o PyBind11 na configuração do seu projeto. Neste exercício, você usa a janela **PowerShell do Desenvolvedor**.

1. Abra a janela **Ferramentas**>**Linha de comando**>**PowerShell do desenvolvedor**.
2. Na janela **PowerShell do desenvolvedor**, instale PyBind11 usando o comando `pip`
`pip install pybind11` ou `py -m pip install pybind11`.

O Visual Studio instala PyBind11 e seus pacotes dependentes.

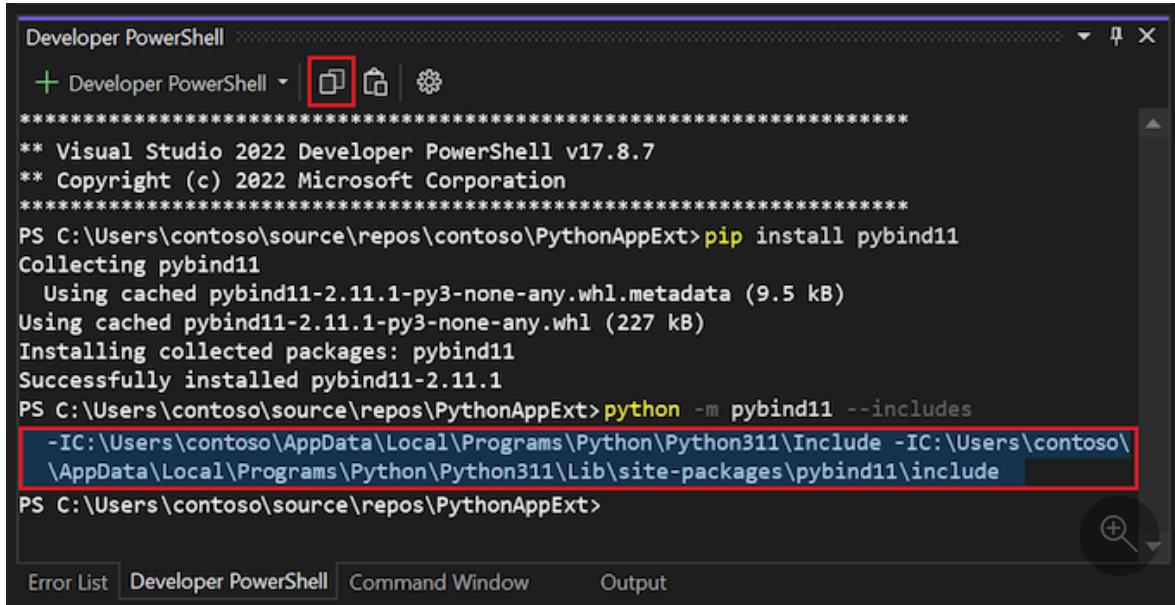
Adicionar os caminhos PyBind11 ao projeto

Após a instalação do PyBind11, você precisa adicionar os caminhos à propriedade **Diretórios de inclusão adicionais** para o projeto.

1. Na janela **PowerShell do desenvolvedor**, execute o comando `python -m pybind11 --includes` ou `py -m pybind11 --includes`.

Essa ação imprime uma lista de caminhos PyBind11 que você deve adicionar às propriedades do projeto.

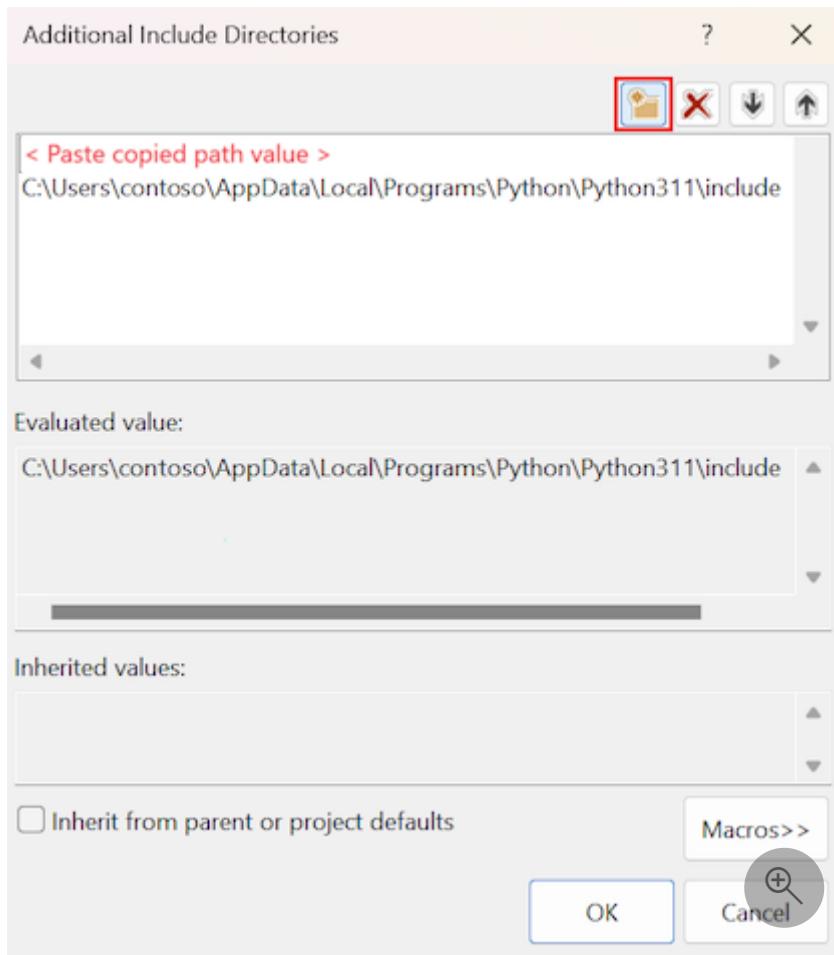
2. Destaque a lista de caminhos na janela e selecione **Copiar** (página dupla) na barra de ferramentas da janela.



```
Developer PowerShell
+ Developer PowerShell | ⌂ ⌂ ⌂
*****
** Visual Studio 2022 Developer PowerShell v17.8.7
** Copyright (c) 2022 Microsoft Corporation
*****
PS C:\Users\contoso\source\repos\contoso\PythonAppExt> pip install pybind11
Collecting pybind11
  Using cached pybind11-2.11.1-py3-none-any.whl.metadata (9.5 kB)
Using cached pybind11-2.11.1-py3-none-any.whl (227 kB)
Installing collected packages: pybind11
Successfully installed pybind11-2.11.1
PS C:\Users\contoso\source\repos\PythonAppExt> python -m pybind11 --includes
-IC:C:\Users\contoso\AppData\Local\Programs\Python\Python311\Include -IC:C:\Users\contoso\
\AppData\Local\Programs\Python\Python311\Lib\site-packages\pybind11\include
PS C:\Users\contoso\source\repos\PythonAppExt>
```

A lista de caminhos concatenados é adicionada a sua área de transferência.

3. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto *superfastcode2* e selecione **Propriedades**.
4. Na parte superior da caixa de diálogo **Páginas de propriedade**, para o campo **Configuração**, selecione **Versão**. (Você poderá ver essa opção com o prefixo **Ativo**.)
5. Na caixa de diálogo, na guia **C/C++>Geral**, expanda o menu suspenso para a propriedade **Diretórios de inclusão adicionais** e selecione **Editar**.
6. Na caixa de diálogo de pop-up, adicione a lista de caminhos copiados:
Repita estes passos para cada caminho na lista concatenada copiada da janela **PowerShell do desenvolvedor**:
 - a. Selecione **Nova Linha** (pasta com símbolo de mais) na barra de ferramentas da caixa de diálogo pop-up.



O Visual Studio adiciona uma linha vazia na parte superior da lista de caminhos e posiciona o cursor no início.

- b. Na linha vazia, cole o caminho PyBind11.

Você também pode selecionar **Mais opções (...)** e usar uma caixa de diálogo pop-up do explorador de arquivos para navegar ao local do caminho.

ⓘ Importante

- Se o caminho contiver o prefixo `-I`, remova o prefixo do caminho.
- Para o Visual Studio reconhecer um caminho, o caminho precisa estar em uma linha separada.

Depois de adicionar um novo caminho, o Visual Studio mostra o caminho confirmado no campo **Valor avaliado**.

7. Selecione **OK** para sair da caixa de diálogo pop-up.
8. Na parte superior da caixa de diálogo **Páginas de propriedade**, passe o mouse sobre o valor para a propriedade **Diretórios de inclusão adicionais** e confirme se os caminhos PyBind11 estão presentes.

9. Selecione **OK** para aplicar as alterações de propriedade.

Atualizar o arquivo `module.cpp`

O último passo é adicionar o arquivo de cabeçalho PyBind11 e o código de macro ao arquivo C++ do projeto.

1. Para o projeto C++ *superfastcode2*, abra o arquivo `module.cpp` no editor de código.
2. Adicione uma instrução na parte superior do arquivo `module.cpp` para incluir o arquivo de cabeçalho `pybind11.h`:

C++

```
#include <pybind11/pybind11.h>
```

3. No final do arquivo `module.cpp`, adicione o código para a macro `PYBIND11_MODULE` para definir o ponto de entrada para a função C++:

C++

```
namespace py = pybind11;

PYBIND11_MODULE(superfastcode2, m) {
    m.def("fast_tanh2", &tanh_impl, R"pbdoc(
        Compute a hyperbolic tangent of a single argument expressed in
        radians.
    )pbdoc");

    #ifdef VERSION_INFO
        m.attr("__version__") = VERSION_INFO;
    #else
        m.attr("__version__") = "dev";
    #endif
}
```

4. Compile o projeto C++ e verifique seu código. Se encontrar erros, confira a próxima seção [Solucionar erros de compilação](#).

Solucionar erros de compilação

Analise as seções a seguir para ver os possíveis problemas que podem causar falha na compilação do módulo C++.

Erro: Não é possível localizar o arquivo do cabeçalho

O Visual Studio retorna uma mensagem de erro como **E1696: Não é possível abrir o arquivo fonte "Python.h"** ou **C1083: Não é possível abrir o arquivo de inclusão: "Python.h": Nenhum arquivo ou diretório.**

Esse erro indica que o compilador não consegue localizar um arquivo de cabeçalho (.h) necessário para seu projeto.

- Para o projeto *superfastcode*, verifique se a propriedade de projeto **C/C++>Geral>Diretórios de inclusão adicionais** contém o caminho para a pasta *incluir* para a instalação do Python. Revise as etapas em [Configurar propriedades do projeto](#).
- Para o projeto *superfastcode2*, verifique se a mesma propriedade do projeto contém o caminho para a pasta *incluir* para a instalação do PyBind11. Revise as etapas [Adicionar os caminhos PyBind11 ao projeto](#).

a obter mais informações sobre como acessar as informações de configuração de instalação do Python, consulte a [Documentação do Python](#).

Erro: não é possível localizar bibliotecas do Python

O Visual Studio retorna um erro indicando que o compilador não pode localizar os arquivos de biblioteca (DLL) necessários para seu projeto.

- Para o projeto C++ (*superfastcode* ou *superfastcode2*), verifique se a propriedade **Vinculador>Geral>Diretórios de biblioteca adicionais** contém o caminho para a pasta *libs* para sua instalação do Python. Revise as etapas em [Configurar propriedades do projeto](#).

a obter mais informações sobre como acessar as informações de configuração de instalação do Python, consulte a [Documentação do Python](#).

Erros do vinculador relacionados à arquitetura de destino

O Visual Studio relata erros do vinculador relacionados à configuração da arquitetura de destino para seu projeto, como x64 ou Win32.

- Para o projeto C++ (*superfastcode* ou *superfastcode2*), altere a configuração de destino para corresponder à sua instalação do Python. Por exemplo, se a configuração de destino do projeto C++ for Win32, mas a instalação do Python for de 64 bits, mude a configuração de destino do projeto C++ para x64.

Testar o código e comparar os resultados

Agora que você tem as DLLs estruturadas como extensões de Python, é possível consultá-las por meio do projeto do Python, importar o módulo e usar seus métodos.

Disponibilize a DLL para o Python

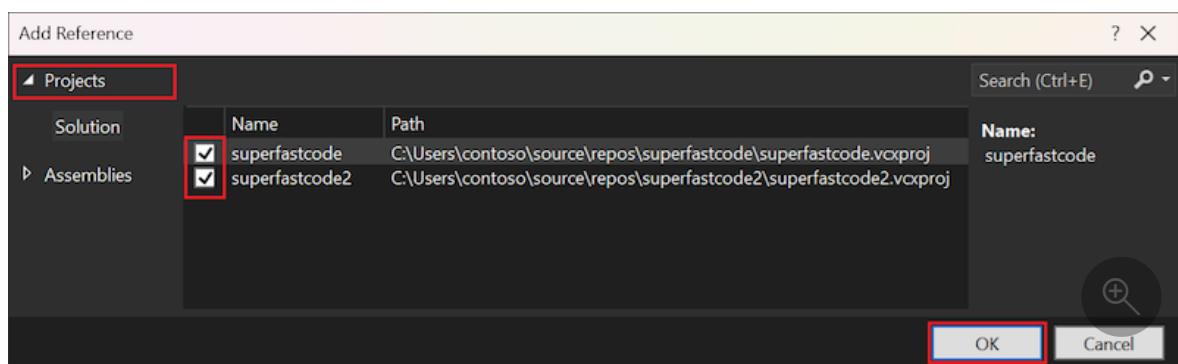
Você pode disponibilizar a DLL para Python de diversas maneiras. Considere estas duas opções:

Se o projeto Python e o projeto C++ estiverem na mesma solução, você poderá usar esta abordagem:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó **Referências** em seu projeto do Python e selecione **Adicionar referência**.

Lembre-se de executar essa ação para seu projeto Python e não para seu projeto C++.

2. Na caixa de diálogo **Adicionar referência**, expanda a guia **Projetos**.
3. Marque as caixas de seleção para os projetos **superfastcode** e **superfastcode2** e selecione **OK**.



Uma abordagem alternativa é instalar o módulo de extensão C++ no seu ambiente Python. Esse método disponibiliza o módulo para outros projetos Python. Para obter mais informações, confira a [documentação do projeto setuptools](#).

Execute estes passos para instalar o módulo de extensão C++ em seu ambiente Python:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nó do projeto e selecione **Adicionar>Novo Item**.
2. Na lista de modelos de arquivo, selecione o **Arquivo C++ (.cpp)**.
3. Insira o **Nome** para o arquivo como *setup.py*, e selecione **Adicionar**.

Lembre-se de inserir o nome do arquivo com a extensão Python (.py). O Visual Studio reconhece o arquivo como código Python, apesar do uso do modelo de arquivo C++.

O Visual Studio abre o novo arquivo no editor de código.

4. Cole o código seguinte no novo arquivo. Escolha a versão do código que corresponda ao seu método de extensão:

- Extensões de CPython (projeto *superfastcode*):

```
Python

from setuptools import setup, Extension

sfc_module = Extension('superfastcode', sources = ['module.cpp'])

setup(
    name='superfastcode',
    version='1.0',
    description='Python Package with superfastcode C++ extension',
    ext_modules=[sfc_module]
)
```

- PyBind11 (projeto *superfastcode2*):

```
Python

from setuptools import setup, Extension
import pybind11

cpp_args = ['-std=c++11', '-stdlib=libc++', '-mmacosx-version-min=10.7']

sfc_module = Extension(
    'superfastcode2',
    sources=['module.cpp'],
    include_dirs=[pybind11.get_include()],
    language='c++',
    extra_compile_args=cpp_args,
)

setup(
    name='superfastcode2',
    version='1.0',
    description='Python package with superfastcode2 C++ extension (PyBind11)',
    ext_modules=[sfc_module],
)
```

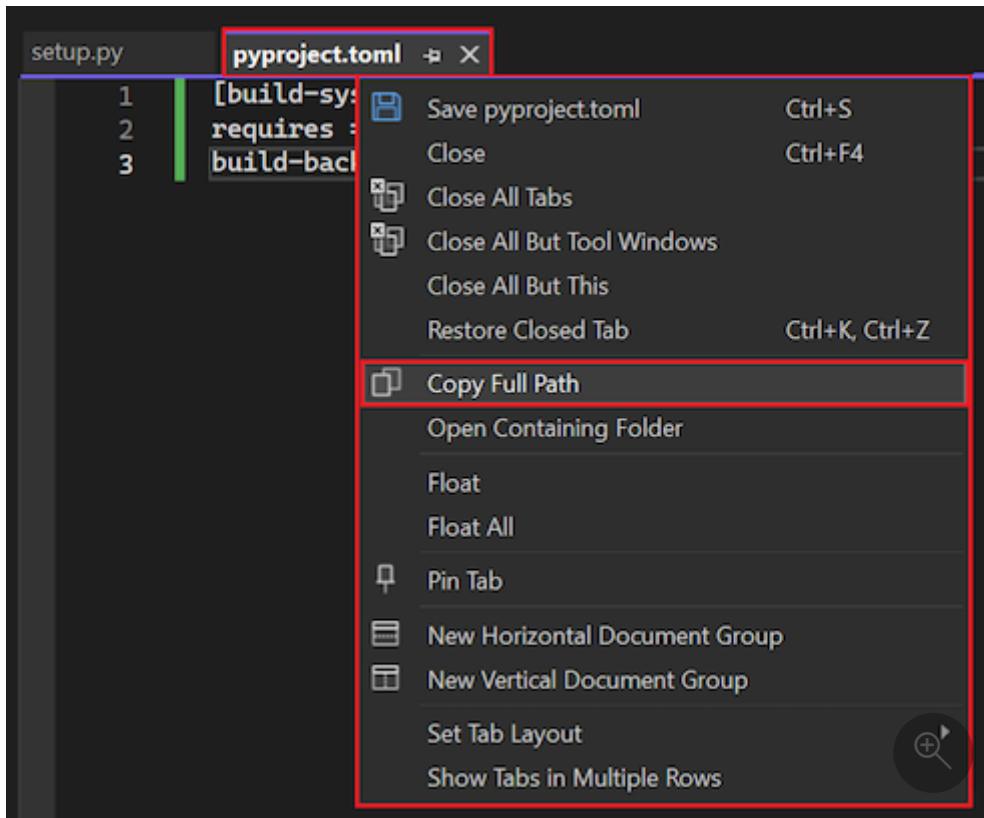
5. No projeto C++, crie um segundo arquivo chamado *pyproject.toml* e cole o seguinte código:

```
toml

[build-system]
requires = ["setuptools", "wheel", "pybind11"]
build-backend = "setuptools.build_meta"
```

O arquivo [TOML](#) (.toml) usa o formato Tom's Obvious, Minimal Language para arquivos de configuração.

6. Para compilar a extensão, clique com o botão direito do mouse no nome de arquivo *pyproject.toml* na guia da janela do código e selecione **Copiar caminho completo**.



Você exclui o nome *pyproject.toml* do caminho antes de usá-lo.

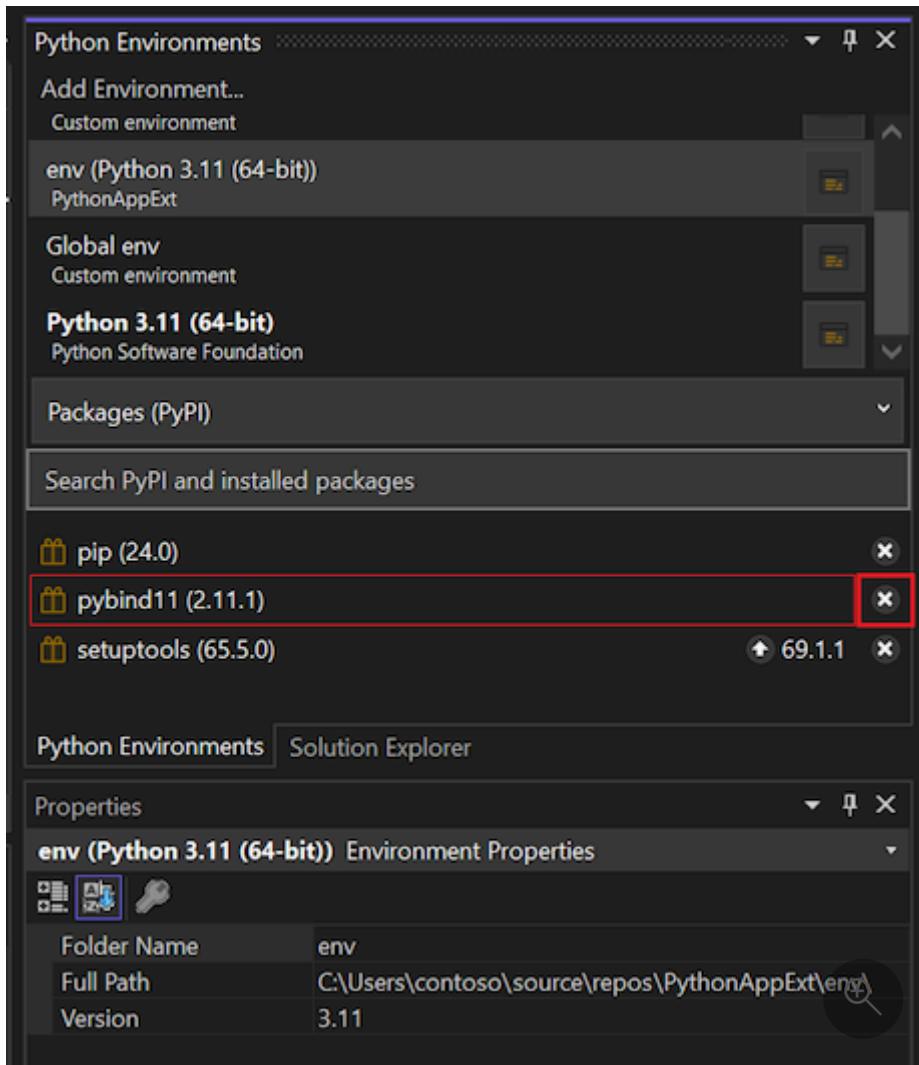
7. No **Gerenciador de soluções**, expanda o nó **Ambientes Python** da solução.

8. Clique com o botão direito no ambiente Python ativo (mostrado em negrito) e selecione **Gerenciar Pacotes Python**.

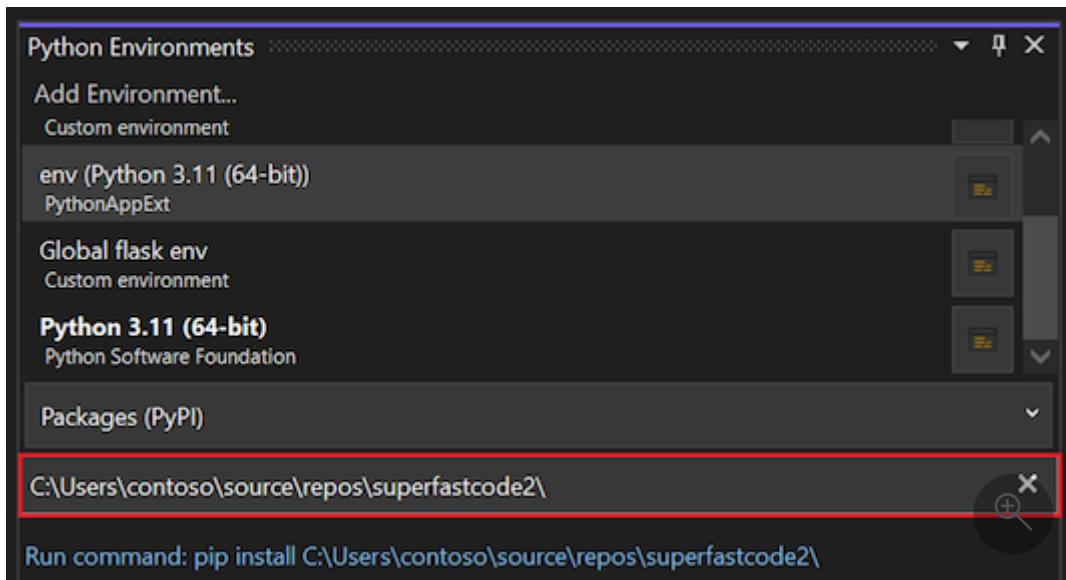
O painel **Ambientes Python** é aberto.

Se o pacote necessário já estiver instalado, você o verá neste painel.

- Antes de continuar, selecione o X do lado do nome do pacote para desinstalá-lo.



9. Na caixa de pesquisa do painel **Ambientes Python**, cole o caminho copiado e exclua o nome de arquivo *pyproject.toml* do final do caminho.



10. Selecione **Enter** para instalar o módulo do local do caminho copiado.

Dica

Se a instalação falhar devido a um erro de permissão, adicione o argumento `-user` ao final do comando e tente a instalação novamente.

Chamar a DLL no Python

Depois de disponibilizar a DLL para o Python, conforme descrito na seção anterior, você estará pronto para chamar as funções `superfastcode.fast_tanh` e `superfastcode2.fast_tanh2` do Python. Depois, você pode comparar o desempenho da função com a implementação do Python.

Siga estes passos para chamar a DLL do módulo de extensão do Python:

1. Abra o arquivo `.py` para seu projeto Python no editor de código.
2. No final do arquivo, adicione o seguinte código para chamar os métodos exportados das DLLs e mostrar sua saída:

Python

```
from superfastcode import fast_tanh
test(lambda d: [fast_tanh(x) for x in d], '[fast_tanh(x) for x in d]
(CPython C++ extension)')

from superfastcode2 import fast_tanh2
test(lambda d: [fast_tanh2(x) for x in d], '[fast_tanh2(x) for x in d]
(PyBind11 C++ extension)')
```

3. Execute o programa Python selecionando **Depurar>Iniciar sem Depuração** ou use o atalho de teclado **Ctrl+F5**.

Observação

Se o comando **Iniciar sem depuração** estiver indisponível, no **Gerenciador de soluções**, clique com o botão direito do mouse no projeto Python e escolha **Definir como Projeto de Inicialização**.

Quando o programa é executado, observe que as rotinas C++ são executadas cerca de 5 a 20 vezes mais rápido que a implementação do Python.

Veja este exemplo de saída típica do programa:

Saída

```
Running benchmarks with COUNT = 500000
[tanh(x) for x in d] (Python implementation) took 0.758 seconds

[fast_tanh(x) for x in d] (CPython C++ extension) took 0.076 seconds

[fast_tanh2(x) for x in d] (PyBind11 C++ extension) took 0.204 seconds
```

4. Tente aumentar a variável `COUNT` para que as diferenças de tempo sejam mais evidentes.

Uma compilação de *Depuração* do módulo C++ também é executada mais lentamente do que uma compilação de *Versão*, pois o build de Depuração é menos otimizado e contém diversas verificações de erro. Tente alternar entre essas configurações de compilação para comparação, mas lembre-se de atualizar as propriedades definidas antes para a configuração de versão.

Aborde a velocidade e a sobrecarga do processo

Na saída, você pode notar que a extensão PyBind11 não é tão rápida quanto a extensão CPython, embora deva ser mais rápida que a implementação de Python pura. O principal motivo para a diferença é por causa do uso do sinalizador [METH_O](#). Esse sinalizador não dá suporte a vários parâmetros, nomes de parâmetros ou argumentos de palavras-chave. O PyBind11 gera um código um pouco mais complexo para fornecer uma interface mais semelhante ao Python aos chamadores. Como o código de teste chama a função 500.000 vezes, os resultados podem amplificar muito essa sobrecarga.

Você pode reduzir ainda mais a sobrecarga movendo o loop `for` para o código nativo Python. Essa abordagem envolve o uso do [protocolo iterador](#) (ou o tipo `py::iterable`) PyBind11 para o [parâmetro de função](#)) para processar cada elemento. Remover as transições repetidas entre Python e C++ é uma maneira eficaz de reduzir o tempo necessário para processar a sequência.

Solucionar os problemas de importação de erros

Se você receber uma mensagem de `ImportError` ao tentar importar seu módulo, poderá resolvê-la de uma das seguintes maneiras:

- Ao compilar por meio de uma referência de projeto, verifique se as propriedades do projeto C++ correspondem ao ambiente Python ativado do projeto Python. Confirme se os mesmos locais de pasta estão em uso para os arquivos *Incluir (.h)* e *Biblioteca (DLL)*.

- Confirme que o arquivo de saída esteja nomeado corretamente, como *superfastcode.pyd*. Um nome ou extensão incorreta evita a importação do arquivo necessário.
- Se você instalou o módulo usando o arquivo *setup.py*, execute o comando `pip` no ambiente `python` ativado no seu projeto Python. Quando você expande o ambiente Python ativo para seu projeto no **Gerenciador de Soluções**, você deve ver uma entrada para o projeto C++, como *superfastcode*.

Depurar o código C++

O Visual Studio é compatível com a depuração de código de Python e C++ juntos. Os passos a seguir demonstram o processo de depuração para o projeto C++ *superfastcode*, mas o processo é o mesmo para o projeto *superfastcode2*.

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto Python e selecione **Propriedades**.
2. No painel **Propriedades**, selecione a guia **Depurar** e a opção **Depurar>Habilitar a depuração de código nativo**.

Dica

Quando você habilitar a depuração de código nativo, a janela de Saída do Python poderá fechar imediatamente depois que o programa terminar sem pausar ou apresentar o prompt **Pressione qualquer tecla para continuar**. Para forçar a pausa e o prompt depois de habilitar a depuração de código nativo, adicione o argumento `-i` ao campo **Executar>Argumentos do interpretador** na guia **Depurar**. Esse argumento coloca o interpretador Python no modo interativo após a execução do código. O programa aguarda você selecionar **Ctrl+Z+Enter** para fechar a janela. Uma abordagem alternativa é adicionar as instruções `import os` e `os.system("pause")` no final do seu programa Python. Esse código duplicará o prompt de pausa original.

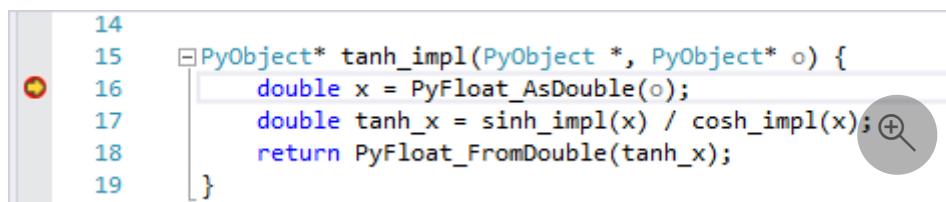
3. Selecione **Arquivo>Salvar** (ou **Ctrl+S**) para salvar as alterações de propriedade.
4. Na barra de ferramentas do Visual Studio, defina a configuração **Build** para **Depurar**.
5. Como o código geralmente demora mais para ser executado no depurador, talvez seja interessante alterar a variável `COUNT` no seu arquivo de projeto Python `.py` para

um valor que seja cerca de cinco vezes menor do que o valor padrão. Por exemplo, altere-o de **500000** para **100000**.

6. No seu código C++, defina um ponto de interrupção na primeira linha do método `tanh_impl`.

7. Inicie o depurador selecionando **Depurar>Iniciar depuração** ou use o atalho de teclado **F5**.

O depurador para quando o código do ponto de interrupção é chamado. Se o ponto de interrupção não for atingido, confirme que a configuração esteja definida como **Depuração** e que você tenha salvado o projeto (o que não acontece automaticamente ao iniciar o depurador).



8. Neste ponto de interrupção, você poderá executar o código C++ em etapas, examinar variáveis e assim por diante. Para obter mais informações sobre esses recursos, consulte [Depurar Python e C++ juntos](#).

Abordagens alternativas

Você pode criar extensões do Python de várias maneiras, conforme descrito na tabela a seguir. As duas primeiras linhas, `CPython` e `PyBind11`, são abordadas neste artigo.

[] Expandir a tabela

| Abordagem | Vintage | Usuários representantes |
|--|---------|---|
| Módulos de extensão do C/C++ para <code>CPython</code> | 1991 | Biblioteca Padrão |
| PyBind11 (recomendado para C++) | 2015 | |
| Cython (recomendado para C) | 2007 | gevent , kivy |
| HPy | 2019 | |
| mypy | 2017 | |
| <code>ctypes</code> | 2003 | oscrypto |
| <code>cffi</code> | 2013 | cryptography , pypy |

| Abordagem | Vintage | Usuários representantes |
|--------------------------------|---------|----------------------------|
| SWIG | 1996 | crfsuite ↗ |
| Boost.Python ↗ | 2002 | |
| cppyy ↗ | 2017 | |

Conteúdo relacionado

- Acesse os arquivos de exemplo completos no GitHub ([python-samples-vs-cpp-extension](#)) ↗
-

Comentários

Esta página foi útil?

 Yes

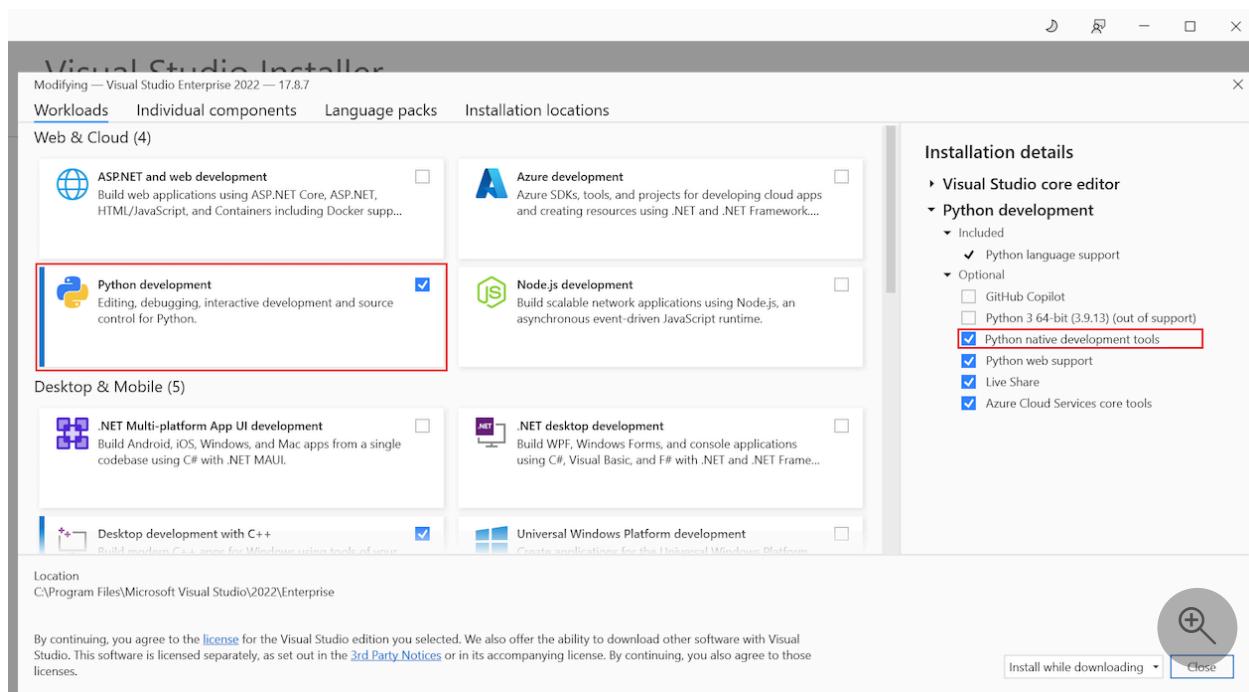
 No

Depurar Python e C++ juntos no Visual Studio

Artigo • 18/04/2024

A maioria dos depuradores Python regulares suportam somente a depuração de código Python, mas é prática comum para desenvolvedores usar Python com C ou C++. Alguns cenários que usam código misto são aplicativos que exigem alta performance ou a capacidade de invocar diretamente APIs de plataforma são frequentemente codificados em Python e C ou C++.

O Visual Studio fornece depuração de modo misto integrada e simultânea para o Python e código C/C++ nativo. O suporte está disponível quando você seleciona a opção de **ferramentas de desenvolvimento nativo Python** para a carga de trabalho **Python Development** no instalador do Visual Studio:



Neste artigo, você explora como trabalhar com estes recursos de depuração de modo misto:

- Pilhas de chamada combinadas
- Etapa entre o Python e o código nativo
- Pontos de interrupção nos dois tipos de código
- Confira as representações de Python de objetos em quadros nativos e vice-versa
- Depuração dentro do contexto do projeto do Python ou C++

The screenshot shows the Visual Studio 2017 IDE during mixed mode debugging. On the left, the Python file `main.py` contains code demonstrating interop between Python and C++. On the right, the C++ file `TestModule.cpp` shows the implementation of a native object. The Locals window shows variables from both sides of the interaction. The Call Stack window traces the execution flow through Python and C++ frames. The Output window shows standard build and debug logs.

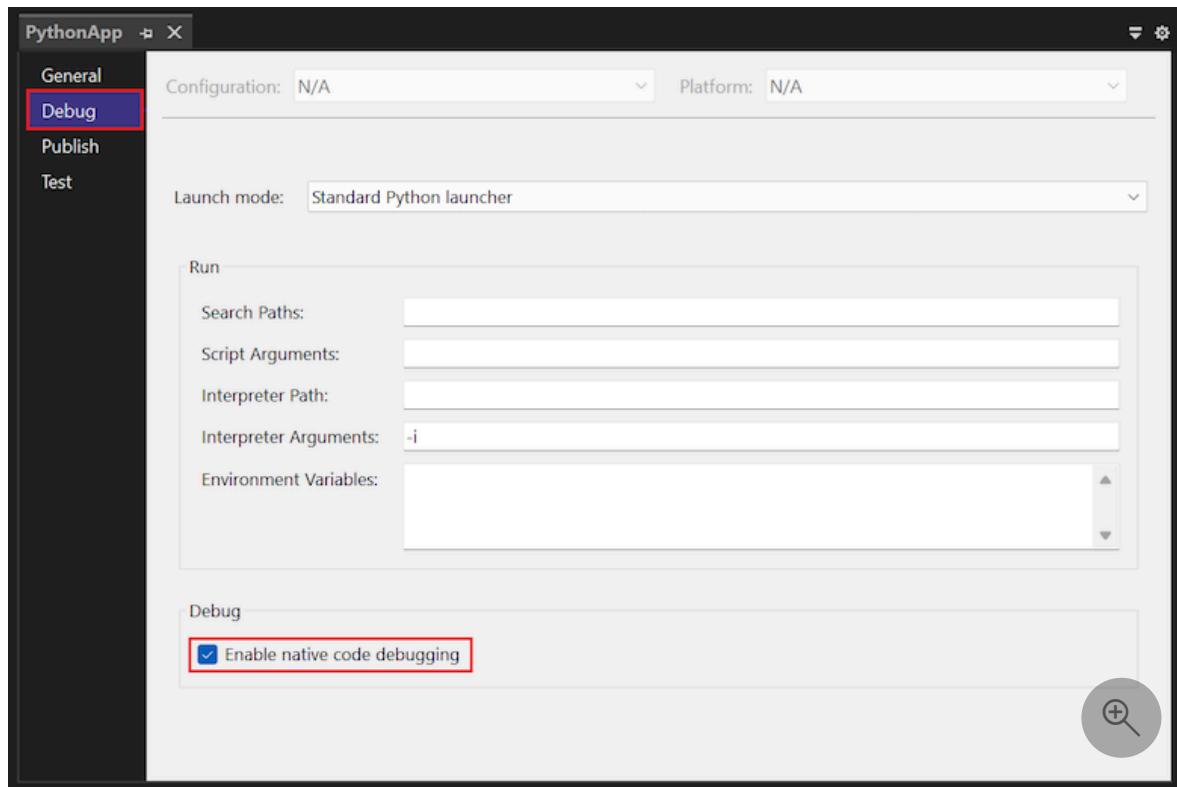
Pré-requisitos

- Visual Studio 2017 e posterior. A depuração de modo misto não está disponível nas Ferramentas Python para Visual Studio 1.x no Visual Studio 2015 e anteriores.
- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Habilitar a depuração de modo misto em um projeto do Python

Os passos a seguir descrevem como habilitar a depuração de modo misto em um projeto Python:

1. No Gerenciador de Soluções, clique com o botão direito do mouse no projeto Python e selecione Propriedades.
2. No painel Propriedades, selecione a guia Depurar e a opção Depurar>Habilitar a depuração de código nativo:



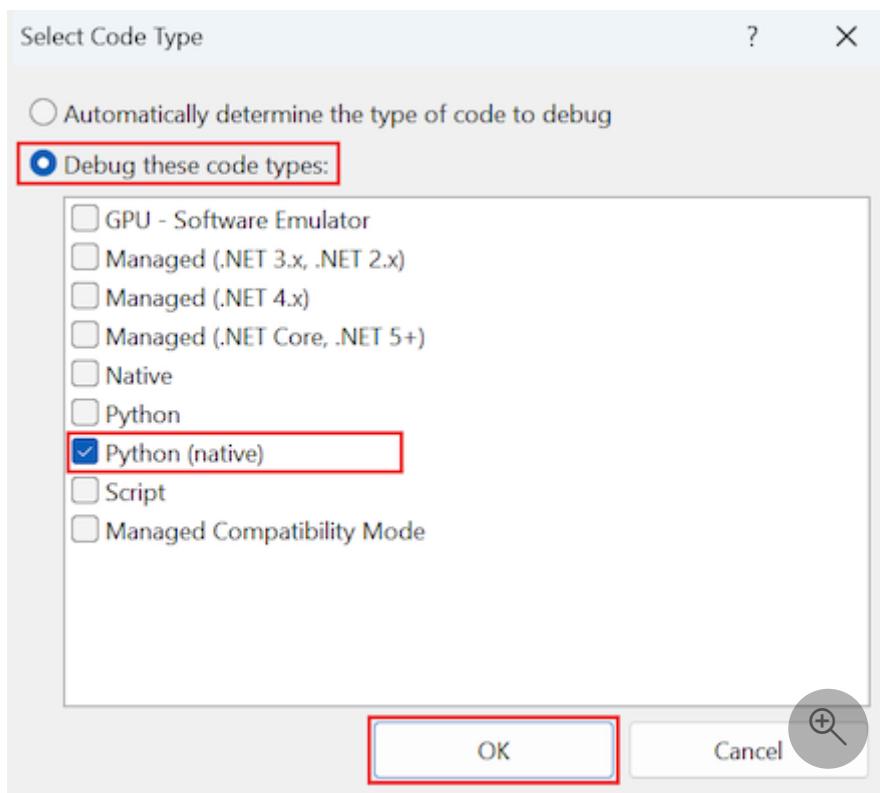
Essa opção habilita o modo misto em todas as sessões de depuração.

Dica

Quando você habilitar a depuração de código nativo, a janela de Saída do Python poderá fechar imediatamente depois que o programa terminar sem pausar ou apresentar o prompt **Pressione qualquer tecla para continuar**. Para forçar a pausa e o prompt depois de habilitar a depuração de código nativo, adicione o argumento `-i` ao campo **Executar>Argumentos do interpretador** na guia **Depurar**. Esse argumento coloca o interpretador Python no modo interativo após a execução do código. O programa aguarda você selecionar **Ctrl+Z+Enter** para fechar a janela.

3. Selecione **Arquivo>Salvar** (ou **Ctrl+S**) para salvar as alterações de propriedade.
4. Para anexar o depurador de modo misto a um processo existente, selecione **Depurar>Anexar ao Processo**. Uma caixa de diálogo aparece.
 - a. Na caixa de diálogo **Anexar ao Processo**, selecione o processo protegido na lista.
 - b. Para o campo **Anexar a**, use a opção **Selecionar** para abrir a caixa de diálogo **Selecionar tipo de código**.
 - c. Na caixa de diálogo **Selecionar tipo de código**, escolha a opção **Depurar estes tipos de código**.

d. Na lista, marque a caixa de seleção **Python (nativo)** e selecione **OK**:



e. Escolha **Anexar** para iniciar o depurador.

As configurações do tipo de código são persistentes. Se você quiser desabilitar a depuração de modo misto e anexar a um processo diferente posteriormente, desmarque a caixa de seleção do tipo de código **Python (nativo)** e marque a caixa de seleção **Tipo de código Nativo**.

Você pode selecionar outros tipos de código além da opção **Nativo**. Por exemplo, se um aplicativo gerenciado hospedar o CPython, que por sua vez usa módulos de extensão nativos, e você quiser depurar os três projetos de código, marque as caixas de seleção **Python**, **Nativo** e **Gerenciado**. Essa abordagem traz uma experiência de depuração unificada, incluindo pilhas de chamadas combinadas e etapas entre os três tempos de execução.

Como trabalhar com ambientes virtuais

Ao usar esse método de depuração de modo misto para ambientes virtuais (venvs), o Python para Windows usa um arquivo stub `python.exe` para venvs que o Visual Studio localiza e carrega como um subprocesso.

- Para o Python 3.8 e posterior, o modo misto não dá suporte à depuração de vários processos. Quando você inicia a sessão da depuração, o subprocesso de stub é depurado em vez do aplicativo. Em cenários de anexação, a solução alternativa é

anexar ao arquivo `python.exe` correto. Quando inicia o aplicativo com depuração (como pelo atalho de teclado F5), você pode criar seu venv usando o comando `C:\Python310-64\python.exe -m venv venv --symlinks`. No comando, coloque sua versão preferida do Python. Por padrão, somente Administradores podem criar links simbólicos no Windows.

- Para versões do Python antes da 3.8, a depuração de modo misto deve funcionar conforme o esperado com venvs.

A execução em um ambiente global não causa esses problemas para nenhuma versão do Python.

Instalar símbolos Python

Ao iniciar a depuração no modo misto pela primeira vez, você poderá ver uma caixa de diálogo **Símbolos Obrigatórios do Python**. Você precisa instalar os símbolos apenas uma vez para qualquer ambiente do Python. Os símbolos serão incluídos automaticamente se você instalar o suporte do Python por meio do Instalador do Visual Studio (Visual Studio 2017 e posterior). Para mais informações, consulte [Instalar símbolos de depuração para interpretadores Python no Visual Studio](#).

Acessar o código-fonte Python

Você pode disponibilizar o código-fonte do Python padrão durante a depuração.

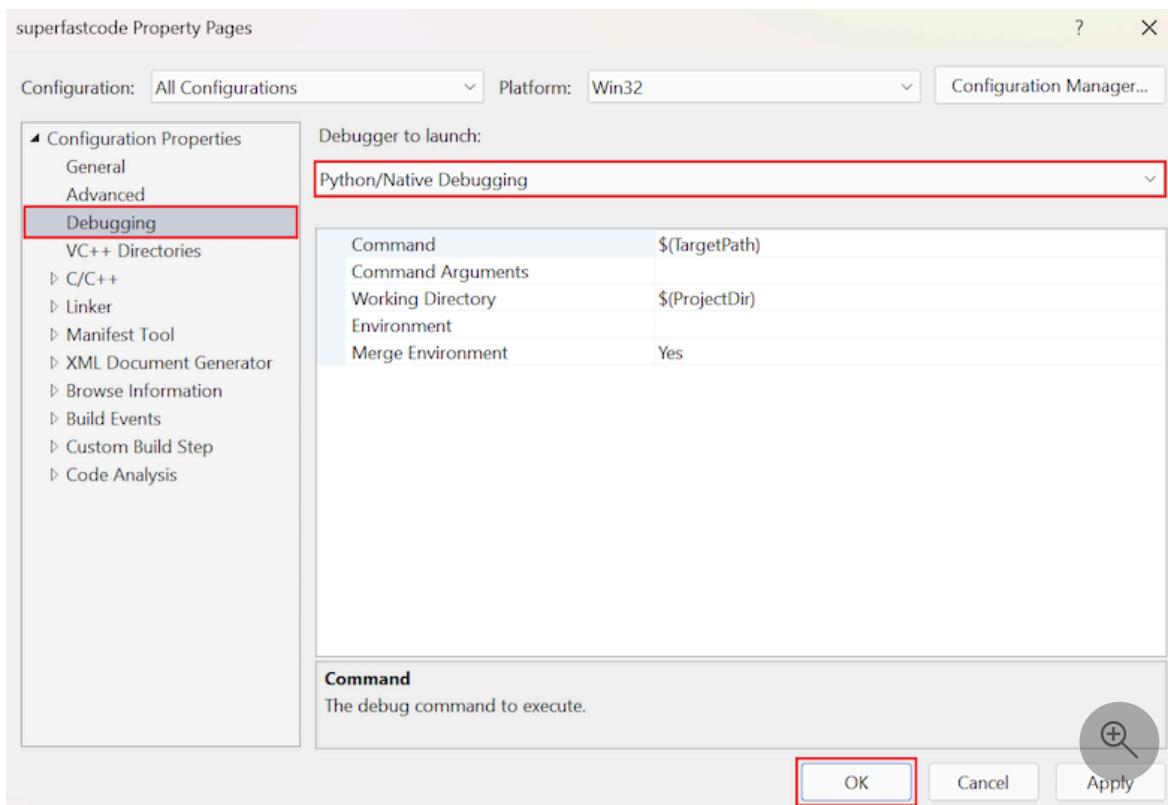
1. Ir para <https://www.python.org/downloads/source/>.
2. Faça o download do arquivo de código-fonte Python apropriado para sua versão e extraia o código para uma pasta.
3. Quando o Visual Studio solicita o local do código-fonte Python, indique para os arquivos específicos na pasta de extração.

Habilitar a depuração de modo misto em um projeto do C/C++

O Visual Studio 2017 versão 15.5 e posterior dá suporte à depuração de modo misto de um projeto C/C++. Um exemplo desse uso é quando você quer [incorporar o Python em outro aplicativo, conforme descrito em python.org](#).

Os passos a seguir descrevem como habilitar a depuração de modo misto para um projeto C/C++:

1. No Gerenciador de Soluções, clique com o botão direito do mouse no projeto C/C++ e selecione Propriedades.
2. No painel Páginas de Propriedades, selecione a guia Propriedades de Configuração>Depuração.
3. Expanda o menu suspenso para a opção Depurador para iniciar e selecione Python/Depuração Nativa.



⚠️ Observação

Se você não tiver a opção Python/Depuração Nativa, primeiro instale as **ferramentas de desenvolvimento nativas do Python** usando o instalador do Visual Studio. A opção de depuração nativa está disponível na carga de trabalho para desenvolvimento do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

4. Selecione Ok para salvar as alterações.

Depurar o iniciador de programas

Ao usar este método, não é possível depurar o Launcher de programa `py.exe` porque ele gera um subprocesso filho `python.exe`. O depurador não é anexado ao subprocesso.

Nesse cenário, a solução alternativa é iniciar o programa `python.exe` diretamente com argumentos, da seguinte maneira:

1. No painel **Páginas de propriedade** para o projeto C/C++, acesse a guia **Propriedades de configuração > Depuração**.
2. Na opção **Comando**, especifique o caminho completo para o arquivo de programa `python.exe`.
3. Especifique os argumentos desejados no campo **Argumentos de comando**.

Anexar o depurador de modo misto

Para o Visual Studio 2017 versão 15.4 e anteriores, a depuração direta de modo misto é ativada somente ao iniciar um projeto Python no Visual Studio. O suporte é limitado porque os projetos C/C++ usam somente o depurador nativo.

Para este cenário, a solução alternativa é anexar o depurador separadamente:

1. Inicie o projeto C++ sem depurar selecionando **Depurar>Iniciar sem depurar** ou use o atalho do teclado **Ctrl+F5**.
2. Para anexar o depurador de modo misto a um processo existente, selecione **Depurar>Anexar ao Processo**. Uma caixa de diálogo aparece.
 - a. Na caixa de diálogo **Anexar ao Processo**, selecione o processo protegido na lista.
 - b. Para o campo **Anexar a**, use a opção **Selecionar** para abrir a caixa de diálogo **Selecionar tipo de código**.
 - c. Na caixa de diálogo **Selecionar tipo de código**, escolha a opção **Depurar estes tipos de código**.
 - d. Na lista, marque a caixa de seleção **Python (nativo)** e selecione **OK**.
 - e. Escolha **Anexar** para iniciar o depurador.

Dica

Você pode introduzir uma pausa ou atraso no aplicativo C++ para garantir que ele não chame o código Python que você quer depurar antes de anexar o depurador.

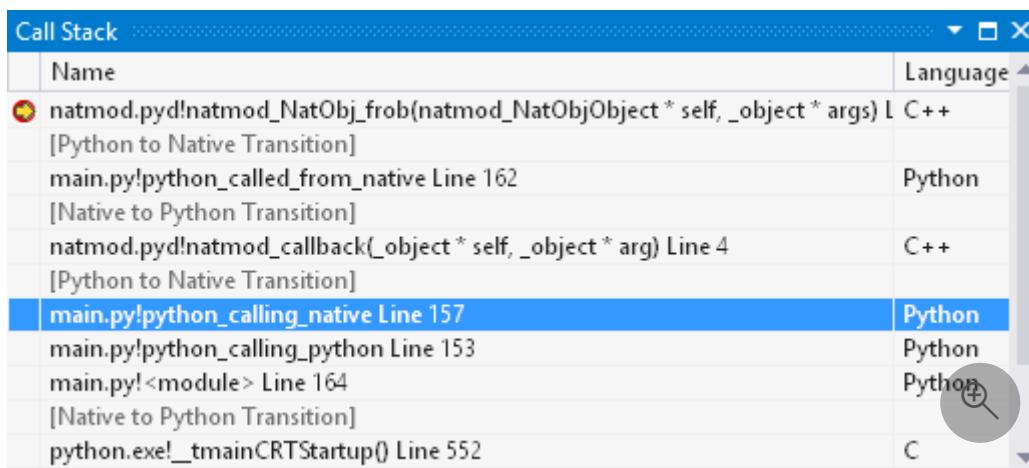
Explore recursos específicos ao modo misto

O Visual Studio tem vários recursos de depuração de modo misto para facilitar a depuração de seu aplicativo:

- Pilha de chamadas combinada
- Etapa entre o Python e o código nativo
- Exibição de valores PyObject no código nativo
- Exibição de valores Nativos no código do Python

Usar uma pilha de chamadas combinada

A janela **Pilha de Chamadas** mostra os registros de ativação nativo e do Python intercalados, com transições marcadas entre os dois:



- Para fazer as transições aparecerem como **[Código Externo]** sem especificar a direção da transição, defina a opção **Ferramentas > Opções > Depuração > Geral > Habilitar Apenas Meu Código**.
- Para tornar ativo qualquer quadro de chamada, clique duas vezes no quadro. Essa ação abre também o código-fonte correspondente, se possível. Se o código-fonte não estiver disponível, o quadro ainda ficará ativo e as variáveis locais poderão ser inspecionadas.

Etapa entre o Python e o código nativo

O Visual Studio oferece os comandos **Intervir** (F11) ou **Sair** (Shift+F11) para permitir que o depurador de modo misto manipule corretamente das alterações entre os tipos de código.

- Quando o Python chama um método de um tipo implementado no C, a intervenção em uma chamada a esse método é interrompida no início da função

nativa que implementa o método.

- Este mesmo comportamento ocorre quando o código nativo chama uma função de API do Python, isso resulta na invocação do código do Python. Intervir em uma chamada para `PyObject_CallObject` em um valor de função que foi originalmente definido no Python é interrompida no início da função do Python.
- Também há suporte para a intervenção do Python para nativo em funções nativas invocadas do Python por meio de [ctypes](#).

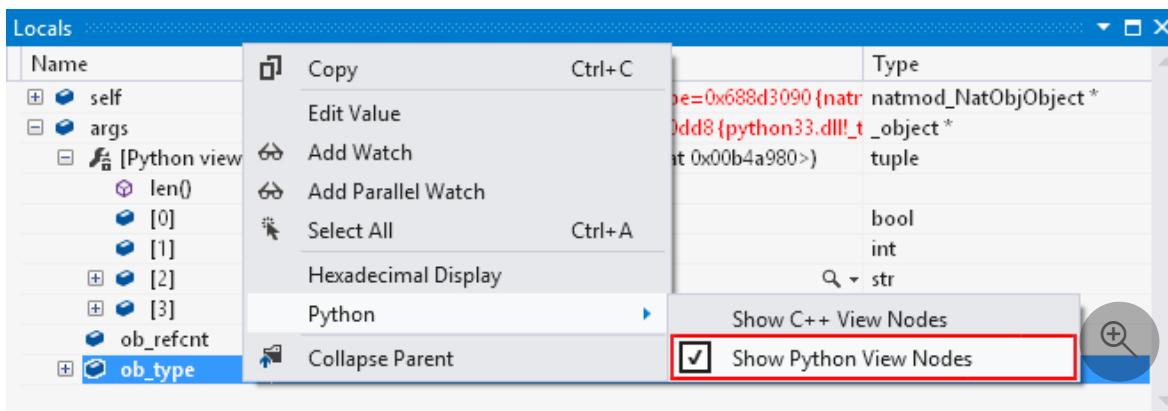
Usar exibição de valores PyObject no código nativo

Quando um quadro nativo (C ou C++) está ativo, suas variáveis locais são mostradas na janela **Locais** do depurador. Os módulos de extensão nativos do Python, muitas dessas variáveis são do tipo `PyObject` (que é um `typedef` de `_object`), ou alguns outros [tipos fundamentais do Python](#). Na depuração de modo misto, esses valores apresentam outro nó filho rotulado [exibição do Python].

- Para ver a representação Python da variável, expanda o nó. A exibição das variáveis é idêntica ao que aparece se uma variável local que faz referência ao mesmo objeto estiver presente em um quadro Python. Os filhos desse nó são editáveis.

| Name | Value | Type |
|-----------------|---|-----------------------|
| + self | 0x00b4a980 {ob_base={ob_refcnt=4 ob_type=0x688d3090 {natr natmod.NatObjObject *}} | natmod.NatObjObject * |
| + args | 0x00bef4b0 {ob_refcnt=1 ob_type=0x1e240dd8 {python33.dll!_typeobject *}} | |
| + [Python view] | (True, 123, 'foo', <natmod.NatObj object at 0x00b4a980>) | tuple |
| len() | 4 | |
| [0] | True | bool |
| [1] | 123 | int |
| [2] | 'foo' | str |
| [3] | <natmod.NatObj object at 0x00b4a980> | natmod.NatObj |
| ob_refcnt | 1 | int |
| ob_type | 0x1e240dd8 {python33.dll!_typeobject PyTuple_Type} {ob_base _typeobject *} | |

- Para desabilitar esse recurso, clique com o botão direito do mouse em qualquer lugar da janela **Locais** e ative/desative a opção de menu **Python>Mostrar Nós de Exibição do Python**:



Tipos C que mostram nós de visualização Python

Os seguintes tipos C mostram nós de [Exibição do Python] (se estiverem habilitados):

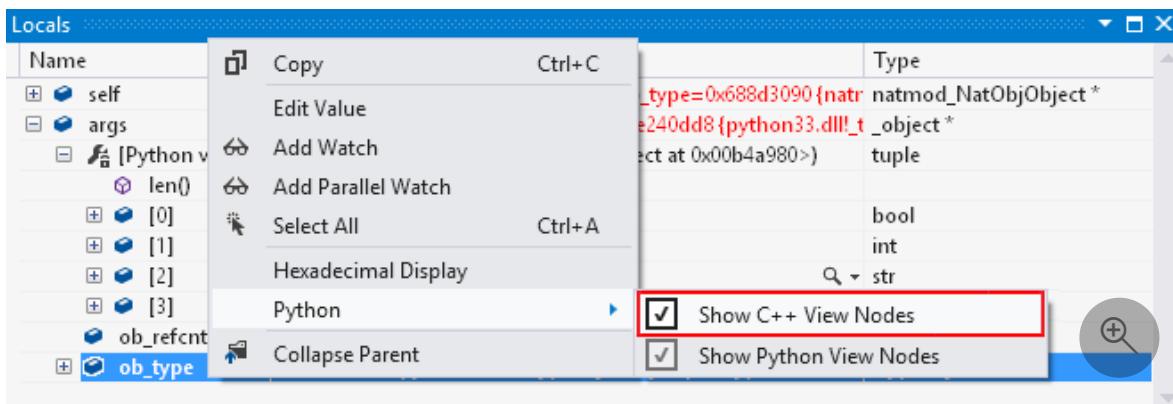
- `PyObject`
- `PyVarObject`
- `PyTypeObject`
- `PyByteArrayObject`
- `PyBytesObject`
- `PyTupleObject`
- `PyListObject`
- `PyDictObject`
- `PySetObject`
- `PyIntObject`
- `PyLongObject`
- `PyFloatObject`
- `PyStringObject`
- `PyUnicodeObject`

A [exibição do Python] não é mostrada automaticamente para tipos criados por sua própria conta. Quando você cria extensões para Python 3.x, essa falha geralmente não é um problema. Qualquer objeto, em última análise, tem um campo `ob_base` de um dos tipos C listados, o que faz com que [visualização Python] apareça.

Exibir valores nativos no código Python

Você pode habilitar uma [Exibição do C++] para valores nativos na janela Locais quando um quadro do Python estiver ativo. Esse recurso não é habilitado por padrão.

- Para ativar o recurso, clique com o botão direito na janela Locais e defina a opção de menu Python>Mostrar Nós de Exibição C++.



- O nó [Exibição do C++] fornece uma representação da estrutura subjacente do C/C++ de um valor, idêntico ao que você veria em um quadro nativo. Ele mostra uma instância de `_longobject` (para a qual `PyLongObject` é um `typedef`) de um inteiro longo do Python e tenta inferir tipos para as classes nativas criadas por conta própria. Os filhos desse nó são editáveis.

| Name | Value | Type |
|---------------|---|------------------------|
| [Globals] | | |
| a_natobj | <natmod.NatObj object at 0x00b4a980> | natmod.NatObj |
| [Python view] | {ob_base={ob_refcnt=4 ob_type=0x688d3090 {natmod.pydl_typeobject natmod.pydlNatmod_NatObj}}} | |
| ob_base | {ob_refcnt=4 ob_type=0x688d3090 {natmod.pydl_typeobject natmod_NatObj natmod.pydl_object}} | natmod.NatObj |
| ob_refcnt | 4 | int |
| ob_type | 0x688d3090 {natmod.pydl_typeobject natmod_NatObjType} {ob_base={ob_l natmod.pydl_typeobject * 0x00b4a980 {ob_refcnt=4 ob_type=0x688d3090 {natmod.pydl_typeobject natmod.pydl_object *}}}} | natmod.pydl_typeobject |
| o | 0x688d20f0 "NatObj" | const char * |
| i | 42 | int |
| s | 42 | int |
| o | <natmod.NatObj object at 0x00b4a980> | natmod.NatObj |
| s | 'NatObj' | const char * |
| mymod | <module object at 0x0bb3fd0> | module |
| n | 123 | int |
| [Python view] | {ob_base={ob_refcnt=14 ob_type=0x1e23e158 {python33.dll!_type _longobject}}} | |
| ob_base | {ob_refcnt=14 ob_type=0x1e23e158 {python33.dll!_typeobject PyVarObject}} | int |
| ob_size | 1 | int |
| ob_digit | 0x1e279dfc {123} | unsigned short[1] |
| natmod | <module object at 0x0bee300> | module |

Se um campo filho de um objeto for do tipo `PyObject`, ou outro tipo compatível, ele terá um nó de representação [Exibição Python] (se essas representações estiverem habilitadas). Esse comportamento possibilita navegar em gráficos de objetos onde os links não são diretamente expostos ao Python.

Ao contrário dos nós [exibição do Python], que usam metadados de objeto do Python para determinar o tipo do objeto, não há nenhum mecanismo similarmente confiável para a [exibição do C++]. Em termos gerais, considerando um valor do Python (ou seja, uma referência `PyObject`), não é possível determinar com confiança qual estrutura do C/C++ está dando suporte a ele. O depurador de modo misto tenta adivinhar esse tipo, observando diversos campos do tipo de objeto (como o `PyTypeObject` referenciado por seu campo `ob_type`) que têm tipos de ponteiro de função. Se um desses ponteiros de função referenciar uma função que pode ser resolvida e essa função tiver um parâmetro

`self` com um tipo mais específico que `PyObject*`, esse tipo será considerado como o tipo de suporte.

Considere o exemplo a seguir, em que o valor `ob_type->tp_init` de um determinado objeto aponta para esta função:

C

```
static int FobObject_init(FobObject* self, PyObject* args, PyObject* kwds) {
    return 0;
}
```

Nesse caso, o depurador poderá deduzir corretamente que o tipo C do objeto é `FobObject`. Se o depurador não conseguir determinar um tipo mais preciso de `tp_init`, ele seguirá para outros campos. Se não for possível deduzir o tipo de nenhum desses campos, o nó [exibição do C++](https://docs.python.org/3/library/_pydev_imps/_pydev_doucumentation.html#the-cpp-expansion-node) apresentará o objeto como uma instância de `PyObject`.

Para obter sempre uma representação útil de tipos criados personalizados, é melhor registrar, pelo menos, uma função especial ao registrar o tipo e usar um parâmetro `self` fortemente tipado. A maioria dos tipos cumpre este requisito naturalmente. Para outros tipos, a inspeção `tp_init` é geralmente a entrada mais conveniente para usar para este fim. Uma implementação fictícia de `tp_init` de um tipo que está presente exclusivamente para habilitar a inferência de tipos do depurador pode apenas retornar zero imediatamente, como no exemplo anterior.

Rever as diferenças da depuração padrão do Python

O depurador de modo misto é diferente do [depurador Python padrão](#). Ele introduz alguns recursos extras, mas não tem alguns recursos relacionados ao Python, como segue:

- Funcionalidades sem suporte incluem pontos de interrupção condicionais, a janela **Interativa de Depuração** e depuração remota multiplataforma.
- A Janela **Imediata**: disponível, mas com um subconjunto limitado de sua funcionalidade, incluindo todas as limitações listadas nesta seção.
- Versões do Python com suporte incluem somente CPython 2.7 e 3.3+.
- Para usar o Python com o Shell do Visual Studio (por exemplo, se você instalá-lo com o instalador integrado), o Visual Studio não consegue abrir projetos C++. Como resultado, a experiência de edição de arquivos C++ é a de um editor de texto básico somente. No entanto, há suporte completo para a depuração do

C/C++ e a depuração de modo misto no Shell com código-fonte, execução em etapas em código nativo e avaliação de expressão do C++ nas janelas do depurador.

- Ao exibir objetos do Python nas janelas **Locais** e **Inspeção** da ferramenta do depurador, o depurador de modo misto mostra somente a estrutura dos objetos. Ele não avalia propriedades automaticamente nem mostra atributos computados. Para coleções, ele mostra apenas os elementos de tipos de coleção interna (`tuple`, `list`, `dict`, `set`). Os tipos de coleção personalizada não são visualizados como coleções, a menos que sejam herdados de algum tipo de coleção interna.
- A avaliação de expressão é tratada como descrito na seção a seguir.

Usar avaliação de expressão

O depurador padrão do Python permite a avaliação de expressões arbitrárias do Python nas janelas **Inspeção** e **Imediata** quando o processo depurado está em pausa em qualquer ponto do código, desde que ele não esteja bloqueado em uma operação de E/S ou em outra chamada do sistema semelhante. Na depuração de modo misto, as expressões arbitrárias podem ser avaliadas somente quando interrompidas no código do Python, depois de um ponto de interrupção ou intervindo no código. As expressões podem ser avaliadas apenas no thread em que o ponto de interrupção ou a operação de intervenção ocorreu.

Quando o depurador pára em código nativo ou em código Python onde as condições descritas não se aplicam, como depois uma operação de "sair" ou em um thread diferente. A avaliação de expressões é limitada ao acesso a variáveis locais e globais no escopo do quadro selecionado atualmente, acessando os seus campos e indexando os tipos de coleção integrados com literais. Por exemplo, a seguinte expressão pode ser avaliada em qualquer contexto (desde que todos os identificadores refiram-se a variáveis e a campos existentes dos tipos apropriados):

```
Python
```

```
foo.bar[0].baz['key']
```

O depurador de modo misto também resolve essas expressões de outra forma. Todas as operações de acesso a membro pesquisam somente os campos que fazem parte diretamente do objeto (como uma entrada em seu `__dict__` ou `__slots__`, ou um campo de um struct nativo que é exposto ao Python por meio de `tp_members`) e ignoram qualquer `__getattr__`, `__getattribute__` ou lógica do descritor. Da mesma forma, todas as operações de indexação ignoram `__getitem__` e acessam as estruturas de dados internas das coleções diretamente.

Por uma questão de consistência, esse esquema de resolução de nomes é usado para todas as expressões que correspondam às restrições para a avaliação de expressão limitada. Esse esquema se aplica independentemente de expressões arbitrárias serem permitidas no ponto de parada atual. Para forçar a semântica correta do Python quando um avaliador completo está disponível, coloque a expressão entre parênteses:

Python

```
(foo.bar[0].baz['key'])
```

Conteúdo relacionado

- [Criar uma extensão do C++ para o Python](#)
- [Instalar símbolos de depuração para interpretadores do Visual Studio](#)

Comentários

Esta página foi útil?

 Yes

 No

Instalar símbolos de depuração para interpretadores do Visual Studio

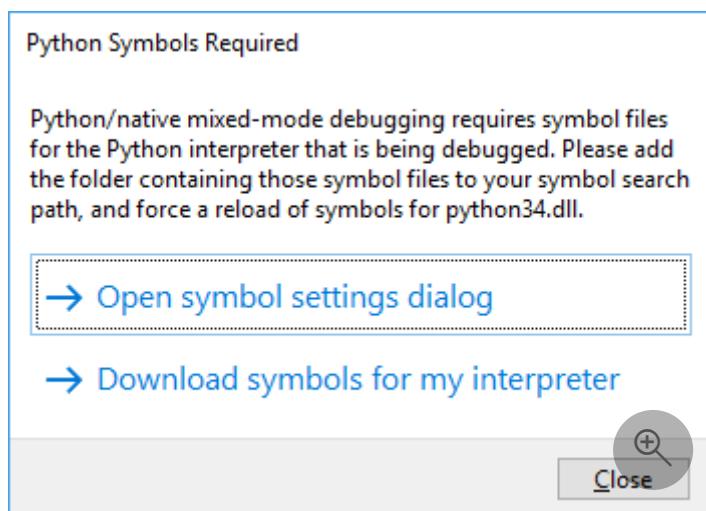
Artigo • 18/04/2024

Este artigo mostra os passos para baixar e integrar símbolos de depuração para interpretadores Python no Visual Studio.

Para fornecer uma experiência de depuração completa, o [depurador de modo misto do Python](#) no Visual Studio precisa de símbolos de depuração para que o interpretador do Python usado analise várias estruturas de dados internas. Os símbolos de depuração são estabelecidos no banco de dados do programa (.pdb). Por exemplo, a biblioteca *python27.dll* exige o arquivo de símbolo *python27.pdb*, a biblioteca *python36.dll* usa o arquivo de símbolo *python36.pdb*, e assim por diante. Cada versão do interpretador também fornece arquivos de símbolo para vários módulos.

- No Visual Studio 2017 e posterior, os interpretadores do Python 3 e do Anaconda 3 instalam automaticamente seus respectivos símbolos e o Visual Studio encontra esses símbolos de forma automática.
- No Visual Studio 2015 e versões anteriores, ou para outros intérpretes, você deve baixar símbolos separadamente e, depois, apontar o Visual Studio para os arquivos.

Quando o Visual Studio detecta símbolos necessários ausentes, uma caixa de diálogo solicita executar uma ação. Normalmente, você vê a caixa de diálogo ao iniciar uma sessão de depuração de modo misto. A caixa de diálogo inclui o link de diálogo **Abrir configurações de símbolo**, que abre a caixa de diálogo **Ferramentas>Opções** para a guia **Depuração>Symbols** juntamente com um link para este artigo de documentação.



Pré-requisitos

- Ter o Visual Studio instalado e compatível com cargas de trabalho do Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).

Verificar a versão do intérprete

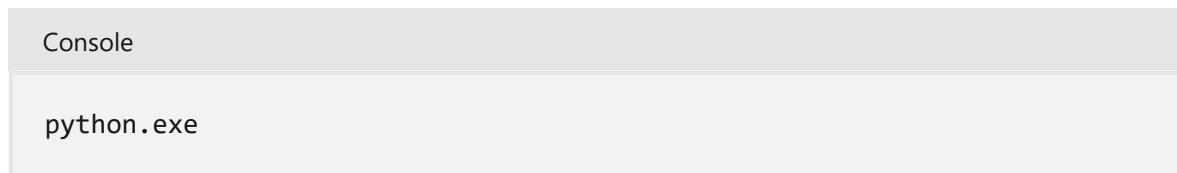
Os símbolos variam entre compilações secundárias do Python e entre compilações de 32 bits e 64 bits. É importante confirmar sua versão e a compilação do Python para garantir que você tenha os símbolos corretos para seu intérprete.

Para verificar qual interpretador Python está usando:

1. No **Gerenciador de Soluções**, expanda o nó **Ambientes do Python** no seu projeto.
2. Encontre o nome do ambiente atual (mostrado em negrito).
3. Clique com o botão direito no nome do ambiente e selecione **Abrir prompt de comando aqui**.

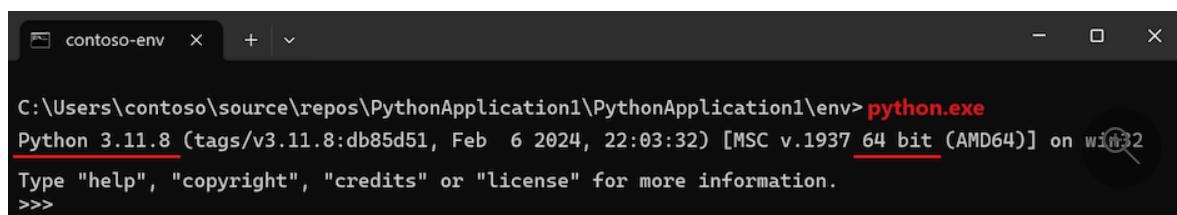
Uma janela de prompt de comando é aberta ao local de instalação do ambiente atual.

4. Abra o python digitando o seguinte comando:



```
Console
python.exe
```

O processo de execução mostra sua versão instalada do Python e indica se ela é de 32 bits ou 64 bits:



```
contoso-env
C:\Users\contoso\source\repos\PythonApplication1\PythonApplication1\env>python.exe
Python 3.11.8 (tags/v3.11.8:db85d51, Feb 6 2024, 22:03:32) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

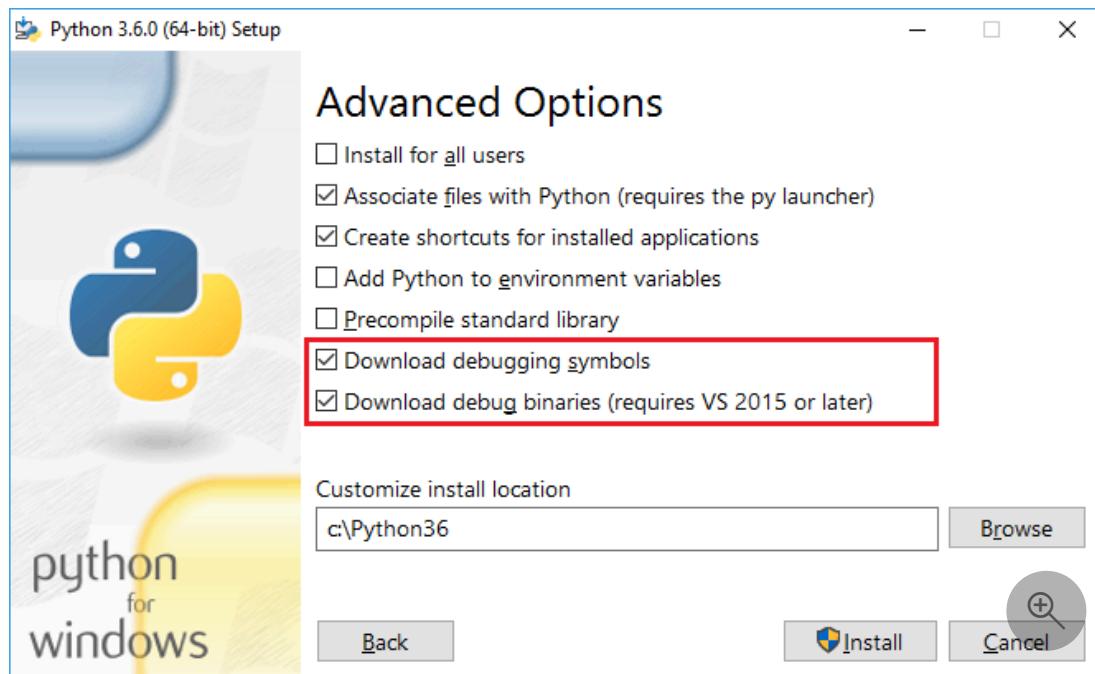
Baixar símbolos

Os passos a seguir descrevem como baixar os símbolos necessários para um interpretador Python.

- Para Python 3.5 e versões posteriores, adquira símbolos de depuração por meio do instalador do Python.

1. Selecione **Instalação personalizada** e **Avançar**.

2. Na página **Opções avançadas**, marque as caixas para **Baixar símbolos de depuração** e **Baixar binários de depuração**:



Os arquivos de símbolo (.pdb) ficam na pasta de instalação raiz. Arquivos de símbolo para módulos individuais também são colocados na pasta *DLLs*.

O Visual Studio encontra esses símbolos automaticamente. Não são necessárias outras etapas.

- No Python 3.4.x e anterior, os símbolos estão disponíveis como arquivos `.zip` de download nas [distribuições oficiais](#) ou no [Enthought Canopy](#).

1. Faça o download do arquivo de símbolo obrigatório.

ⓘ Importante

Lembre-se de selecionar o arquivo de símbolo que corresponde à sua versão instalada do Python e compilação (32 bits ou 64 bits).

2. Extraia os arquivos de símbolo em uma pasta local dentro da pasta Python, como *Símbolos*.
3. Depois de extrair os arquivos, a próxima etapa é apontar o [Visual Studio](#) aos *símbolos*.

- Para outras distribuições do Python de terceiros, como o ActiveState Python, será necessário entrar em contato com os autores dessa distribuição e solicitar que eles forneçam símbolos.

O WinPython incorpora o interpretador Python padrão sem alterações. Você pode usar os símbolos da distribuição oficial do WinPython para o número de versão correspondente.

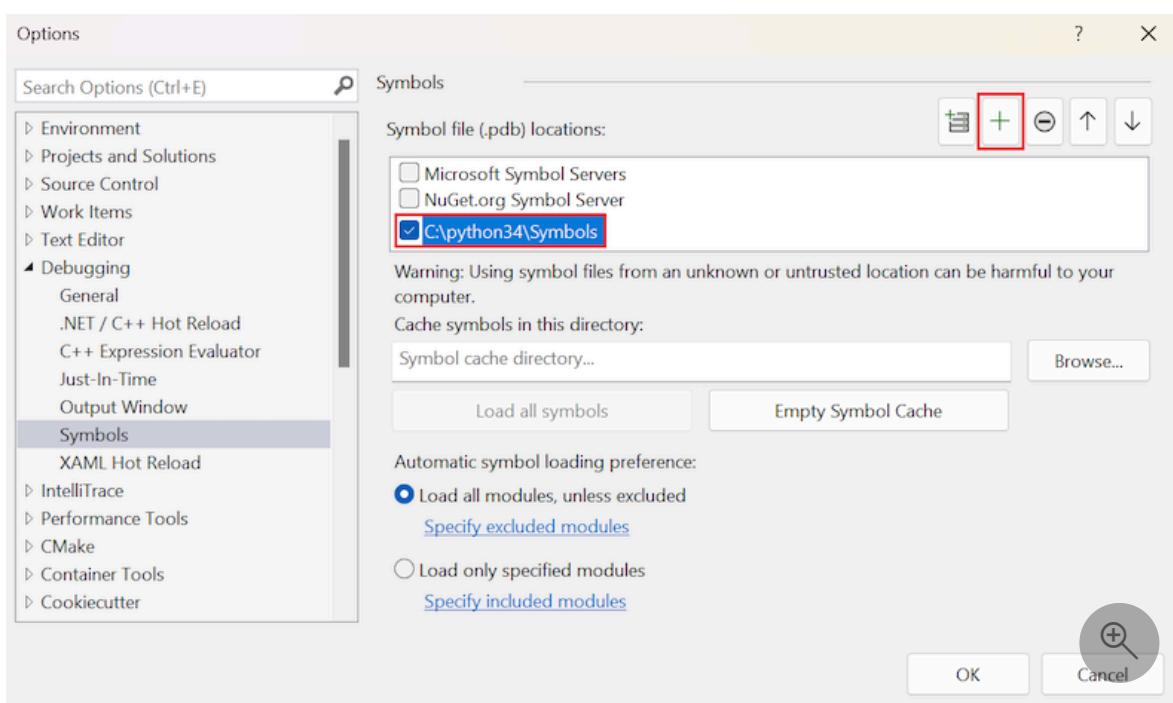
Apontar o Visual Studio para os símbolos

Se os símbolos tiverem sido baixados separadamente, siga estes passos para que o Visual Studio os reconheça.

① Observação

Se os símbolos foram instalados usando o Python 3.5 ou de um instalador posterior, o Visual Studio os encontra automaticamente. Você não precisará concluir os passos nesta seção.

1. Selecione **Ferramentas>Opções** e abra a guia **Depuração>Símbolos**.
2. Selecione **Adicionar** (símbolo de mais) na barra de ferramentas.
3. Insira o caminho da pasta em que você extraiu os símbolos baixados. Esse local é onde fica o arquivo `python.pdb`, como `c:\python34\Symbols`, conforme mostrado na imagem a seguir.



4. Selecione OK.

Durante uma sessão de depuração, o Visual Studio também pode solicitar o local de um arquivo de origem para o interpretador de Python. Se tiver baixado os arquivos de origem, como de python.org/downloads/, você pode apontar o Visual Studio para os arquivos baixados.

Opções de cache de símbolos

A caixa de diálogo **Ferramentas > Opções, Depuração > Símbolos** também contém opções para configurar o cache de símbolos. O Visual Studio usa funcionalidades de cache de símbolo para criar um cache local de símbolos obtidos de uma fonte online.

Essas funcionalidades não serão necessárias com os símbolos do interpretador de Python, pois os símbolos já estão presentes localmente. Para obter mais informações, consulte [Especificar símbolos e arquivos de origem no depurador do Visual Studio](#).

Acessar downloads para distribuições oficiais

A tabela a seguir mostra informações de download para versões oficiais do Python.

 Expandir a tabela

| Versão do Python | Downloads |
|---------------------------|--|
| 3.5 e versões posteriores | Instale símbolos por meio do instalador do Python. |
| 3.4.4 | 32 bits - 64 bits |
| 3.4.3 | 32 bits - 64 bits |
| 3.4.2 | 32 bits - 64 bits |
| 3.4.1 | 32 bits - 64 bits |
| 3.4.0 | 32 bits - 64 bits |
| 3.3.5 | 32 bits - 64 bits |
| 3.3.4 | 32 bits - 64 bits |
| 3.3.3 | 32 bits - 64 bits |
| 3.3.2 | 32 bits - 64 bits |
| 3.3.1 | 32 bits - 64 bits |

| Versão do Python | Downloads |
|------------------|---------------------------------------|
| 3.3.0 | 32 bits ↗ - 64 bits ↗ |
| 2.7.18 | 32 bits ↗ - 64 bits ↗ |
| 2.7.17 | 32 bits ↗ - 64 bits ↗ |
| 2.7.16 | 32 bits ↗ - 64 bits ↗ |
| 2.7.15 | 32 bits ↗ - 64 bits ↗ |
| 2.7.14 | 32 bits ↗ - 64 bits ↗ |
| 2.7.13 | 32 bits ↗ - 64 bits ↗ |
| 2.7.12 | 32 bits ↗ - 64 bits ↗ |
| 2.7.11 | 32 bits ↗ - 64 bits ↗ |
| 2.7.10 | 32 bits ↗ - 64 bits ↗ |
| 2.7.9 | 32 bits ↗ - 64 bits ↗ |
| 2.7.8 | 32 bits ↗ - 64 bits ↗ |
| 2.7.7 | 32 bits ↗ - 64 bits ↗ |
| 2.7.6 | 32 bits ↗ - 64 bits ↗ |
| 2.7.5 | 32 bits ↗ - 64 bits ↗ |
| 2.7.4 | 32 bits ↗ - 64 bits ↗ |
| 2.7.3 | 32 bits ↗ - 64 bits ↗ |
| 2.7.2 | 32 bits ↗ - 64 bits ↗ |
| 2.7.1 | 32 bits ↗ - 64 bits ↗ |

Usar símbolos do Enthought Canopy

O Enthought Canopy fornece símbolos de depuração para seus binários a partir da versão 1.2. Esses símbolos são instalados automaticamente com a distribuição.

- Para usar os símbolos, adicione manualmente a pasta que contém os símbolos ao caminho do símbolo, conforme descrito em [Apontar o Visual Studio para os símbolos](#).

Para uma instalação típica por usuário do Canopy, os símbolos ficam nestas pastas:

- Versão 64 bits: %UserProfile%\AppData\Local\Enthought\Canopy\User\Scripts
- Versão 32 bits: %UserProfile%\AppData\Local\Enthought\Canopy32\User\Scripts

O Enthought Canopy 1.1 e anteriores e o Enthought Python Distribution (EPD) não fornecem símbolos de interpretador. Essas versões não são compatíveis com a depuração de modo misto.

Conteúdo relacionado

- Especificar símbolos e arquivos de origem no depurador do Visual Studio
 - Depurar Python e C++ juntos (depuração de modo misto)
-

Comentários

Esta página foi útil?

 Yes

 No

Criação de perfil para o código Python no Visual Studio

Artigo • 23/04/2024

O Visual Studio fornece recursos de criação de perfil para aplicativos em Python quando você realiza a depuração do código. O **Depurador** do Visual Studio possibilita que você examine o código para verificar variáveis, analisar o estado do programa, resolver possíveis problemas, e assim por diante. Os recursos de criação de perfil do Visual Studio fornecem informações sobre os tempos de execução para o programa. É possível usar essas informações para auxiliar na identificação de problemas de desempenho no código.

A criação de perfil está disponível para o Python 3.9 e para versões anteriores ao usar um interpretador baseado em CPython.

Pré-requisitos

- Instalação do Visual Studio no Windows com suporte para cargas de trabalho em Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Um interpretador baseado em CPython. O [CPython ↗](#) é o interpretador “nativo” e mais usado e está disponível em versões de 32 bits e de 64 bits (a recomendação é para usar 32 bits). Para obter mais informações, confira [Instalar interpretadores do Python](#).
- Um [projeto em Python](#) com código ou uma [pasta com código Python](#).

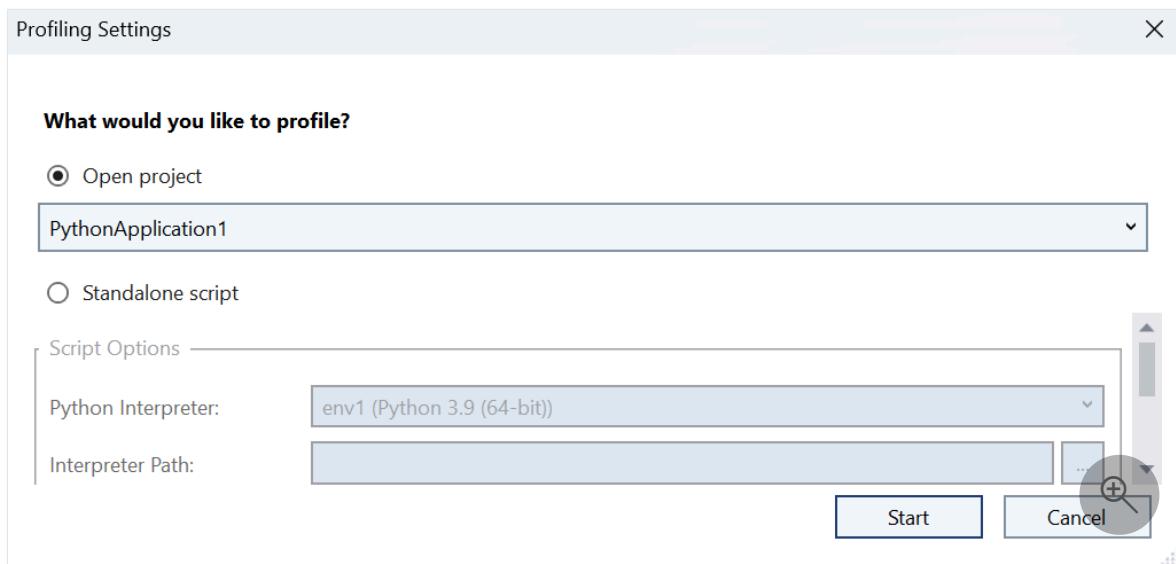
Ainda não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis ↗](#).

Usar o criador de perfil com o interpretador baseado em CPython

Quando você analisa um aplicativo do Python, o Visual Studio coleta dados do tempo de vida do processo.

Siga estas etapas para começar a trabalhar com os recursos de criação de perfil no Visual Studio:

1. No Visual Studio, abra o arquivo de código Python.
2. Confirme se o ambiente atual para o código Python corresponde a um interpretador baseado em CPython. É possível verificar o interpretador selecionado na [janela Ambientes do Python](#).
3. Na barra de ferramentas principal, selecione **Depurar>Iniciar a Criação de Perfil do Python**. O Visual Studio abre a caixa de diálogo **Configurações de Criação de Perfil**:



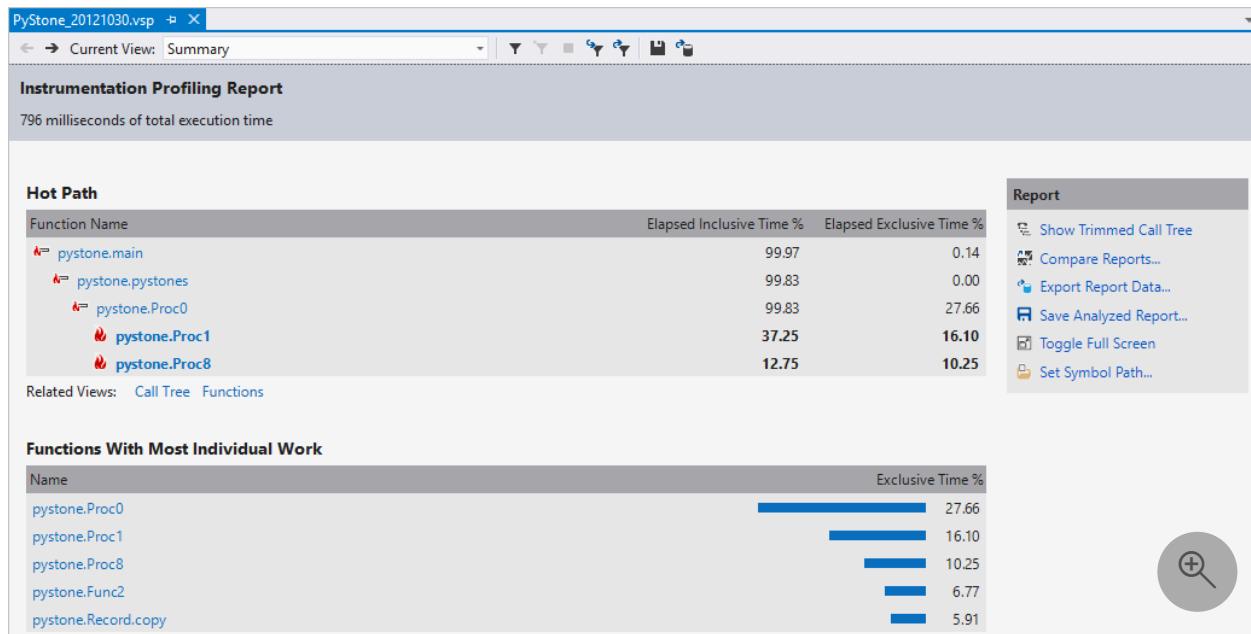
4. Na caixa de diálogo **Configurações de Criação de Perfil**, escolha o arquivo de código ou o código do projeto cujo perfil você deseja criar:
 - Para criar o perfil completo do código do projeto:
 - a. Selecione **Abrir projeto**.
 - b. Selecione o projeto usando a lista suspensa. A lista mostra todos os projetos na solução Visual Studio atual.
 - Para criar o perfil de um arquivo específico:
 - a. Selecione **Script autônomo**.
 - b. Selecione o **Interpretador do Python** usando a lista suspensa ou navegue até a localização. Para especificar um interpretador não listado, escolha **Outros** na lista suspensa e, em seguida, especifique o **Caminho do Interpretador**.
 - c. Identifique o arquivo de **Script** ou navegue até a localização.

d. Especifique o Diretório de Trabalho ou navegue até a localização.

e. Especifique os Argumentos da Linha de Comando para o script.

5. Selecione Iniciar.

Ocorre a execução do criador de perfil e um relatório de desempenho é aberto no Visual Studio. É possível realizar a revisão do relatório para explorar como o tempo é utilizado em seu aplicativo:



Usar o criador de perfil com o IronPython

O IronPython é uma implementação .NET do Python que está disponível nas versões de 32 bits e de 64 bits. O IronPython não é um interpretador baseado em CPython. O Visual Studio oferece suporte à depuração padrão do Python para o IronPython, mas não aos recursos de criação de perfil.

Para projetos em IronPython, é possível usar o criador de perfil do Visual Studio .NET. Realize a execução do comando `ipy.exe`, de forma direta, como o aplicativo de destino com os argumentos apropriados para iniciar o script de inicialização. Na linha de comando, inclua o argumento `-X:Debug` para garantir que todo o código Python seja depurado e passe pelo criador de perfil. Esse argumento gera um relatório de desempenho que inclui o tempo utilizado no runtime do IronPython e seu código. O código é identificado ao usar *nomes desconfigurados*.

O IronPython disponibiliza algumas criações de perfil internas, mas atualmente não há nenhum visualizador funcional. Para obter mais informações, confira [An IronPython Profiler](#) (postagem no blog) e [Debugging and Profiling](#) na documentação do IronPython.

Conteúdo relacionado

- [Instalar interpretadores do Python](#)
 - [Tutorial: Run code in the Debugger in Visual Studio](#)
-

Comentários

Esta página foi útil?

 Yes

 No

Gravação de testes de unidade para o Python com o Gerenciador de Testes no Visual Studio

Artigo • 23/04/2024

Testes de unidade são partes do código que testam outras unidades de código em um aplicativo, normalmente, funções isoladas, classes e assim por diante. Quando um aplicativo passa em todos os testes de unidade, você pode ter certeza de que, no mínimo, a funcionalidade do programa de baixo nível está correta.

O Python usa testes de unidade extensivamente para validar cenários durante a criação de um programa. O suporte do Python no Visual Studio inclui a descoberta, a execução e a depuração de testes de unidade no contexto do processo de desenvolvimento, sem precisar executar os testes separadamente.

Este artigo fornece uma breve descrição das funcionalidades de teste de unidade no Visual Studio com o Python. Para obter mais informações sobre testes de unidade em geral, consulte [Executar um teste de unidade no código](#).

Pré-requisitos

- Instalação do Visual Studio no Windows com suporte para cargas de trabalho em Python. Para obter mais informações, confira [Instalar o suporte ao Python no Visual Studio](#).
- Um [projeto em Python](#) com código ou uma [pasta com código Python](#).

Não há suporte ao Visual Studio para Mac. Para obter mais informações, consulte [O que está acontecendo com o Visual Studio para Mac?](#) O Visual Studio Code no Windows, no Mac e no Linux [funciona perfeitamente com o Python por meio das extensões disponíveis](#).

Seleção da estrutura de teste para um projeto em Python

O Visual Studio oferece suporte a duas estruturas de teste para Python: [unittest](#) e [pytest](#) (que estão disponíveis na versão 16.3 e em versões posteriores do Visual

Studio 2019). Por padrão, nenhuma estrutura é selecionada quando você cria um projeto do Python.

Siga estas etapas para realizar a seleção da estrutura de teste para o projeto em Python:

1. No **Gerenciador de Soluções**, clique com o botão direito do mouse no nome do projeto e selecione **Propriedades**.
2. No painel **Propriedades** do projeto, selecione a guia **Teste** e escolha o seu tipo de **Estrutura de teste**:
 - Para a estrutura **unittest**, o Visual Studio atribui o **Diretório raiz** do projeto para a detecção de testes. O valor padrão é `.`, mas é possível especificar uma localização diferente ao definir as configurações do projeto. Além disso, é possível especificar uma ou mais sequências para o nome do arquivo de teste **Padrão**, como `test*.py`, `test_*.py`.
 - Para a estrutura **pytest**, as opções de teste, como a localização de teste e os padrões do nome do arquivo, são especificadas ao usar o arquivo de configuração `.ini` do pytest padrão. Por padrão, a pasta de espaço de trabalho ou de projeto é usada como a localização. O padrão do nome do arquivo padrão inclui `test_*py` e `*_test.py`. Para saber mais, consulte a [documentação de referência pytest](#).

(!) Observação

Ao definir o padrão do nome do arquivo, lembre-se de que caracteres especiais como o sublinhado (`_`) não correspondem ao caractere curinga (`*`). Se desejar usar caracteres especiais no nome do arquivo, especifique esses caracteres na definição do padrão, como `test_*py`.

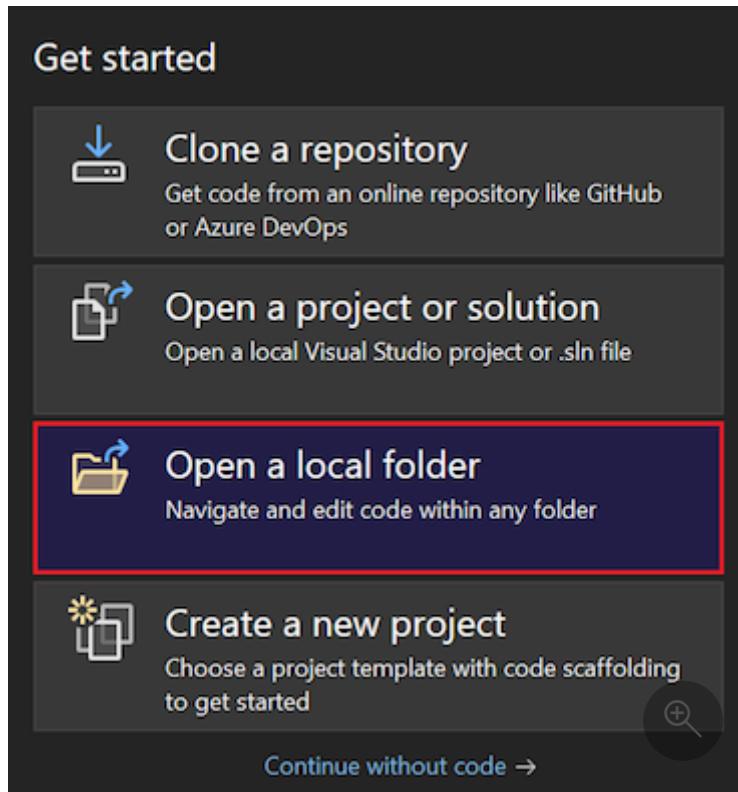
3. Para salvar a seleção e as configurações de estrutura, é possível usar o atalho de teclado **Ctrl+S**.

Após configurar a estrutura, o Visual Studio inicia a detecção de testes e abre o [Gerenciador de Testes](#).

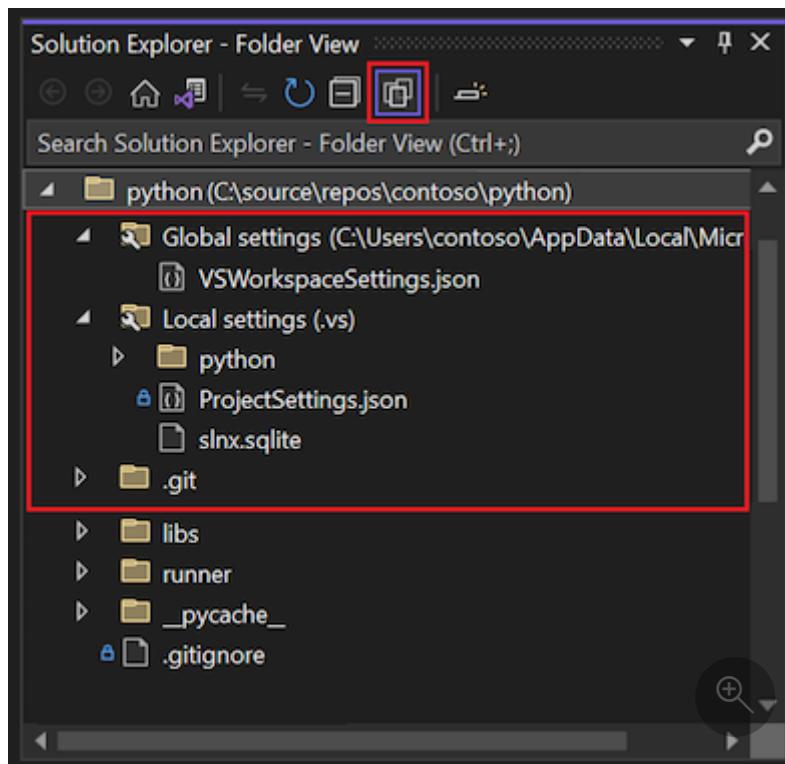
Configurar o teste para Python sem um projeto

O Visual Studio permite que você execute e teste o código Python existente sem um projeto, [abrindo uma pasta](#) com código Python. Nesse cenário, você precisa usar um arquivo `PythonSettings.json` para configurar os testes.

1. Abra o código Python existente ao usar a opção **Abrir uma Pasta Local**:



2. Ao abrir uma pasta do Python, o Visual Studio cria várias pastas ocultas para gerenciar as configurações relacionadas ao programa. Para visualizar essas pastas (e outros arquivos e pastas ocultos, como a pasta `.git`) no **Gerenciador de Soluções**, selecione a opção **Mostrar Todos os Arquivos**:



3. No **Gerenciador de Soluções**, expanda a pasta *Configurações Locais* e realize um clique duplo no arquivo `PythonSettings.json` para abri-lo no editor.

(!) Observação

A maioria das configurações mostra dois arquivos de configurações: *PythonSettings.json* e *ProjectSettings.json*. Para esse exercício, você precisa modificar o arquivo *PythonSettings.json*.

Caso não veja o arquivo *PythonSettings.json* na pasta *Configurações Locais*, você poderá criá-lo de forma manual:

- a. Clique com o botão direito do mouse na pasta *Configurações Locais* e selecione **Adicionar>Novo Arquivo**.
- b. Nomeie o arquivo *PythonSettings.json* e selecione **Enter** para salvar as alterações.

O Visual Studio abre automaticamente o novo arquivo no editor.

4. No arquivo *PythonSettings.json*, adicione o código apresentado a seguir para definir `TestFramework`. Defina o valor da estrutura como `pytest` ou `unittest` com base na estrutura de teste desejada:

JSON

```
{  
    "TestFramework": "unittest",  
    "UnitTestRootDirectory": "testing",  
    "UnitTestPattern": "test_*.py"  
}
```

- Para a estrutura `unittest`, se você não definir valores específicos para as configurações `UnitTestRootDirectory` e `UnitTestPattern` no arquivo *PythonSettings.json*, o Visual Studio adicionará automaticamente esses campos com os valores padrão de `.` e `test*.py`, respectivamente.
- Para a estrutura `pytest`, as opções de configuração são sempre especificadas no arquivo de configuração `.ini` do pytest, e não nas configurações do Visual Studio.

5. Se o programa em Python contém uma pasta `src` separada da pasta que contém os testes, especifique o caminho para a pasta `src` com a configuração `SearchPaths` no arquivo *PythonSettings.json*:

JSON

```
"SearchPaths": [".\\src"]
```

6. Salve as alterações para o arquivo *PythonSettings.json*.

Após configurar a estrutura, o Visual Studio inicia a detecção de testes para a estrutura especificada. É possível acessar o teste no [Gerenciador de Testes](#).

Adição e descoberta de testes

Por padrão, o Visual Studio identifica **unittest** e **pytest** como métodos cujos nomes começam com `test`.

Para visualizar como o Visual Studio inicia a detecção de testes, siga estas etapas:

1. Abra um [projeto em Python](#) no Visual Studio.
2. Configure as **Propriedades** da estrutura de teste para o projeto, conforme descrito em [Seleção da estrutura de teste para um projeto em Python](#).
3. No **Gerenciador de Soluções**, clique com o botão direito do mouse no projeto e selecione **Adicionar>Novo Item**.
 - a. Na caixa de diálogo **Adicionar Novo Item**, selecione o tipo de arquivo **Teste de Unidade em Python**.
 - b. Insira um nome do arquivo que satisfaça a definição do **Padrão** especificada para as **Propriedades** do projeto.
 - c. Selecione **Adicionar**.
4. O Visual Studio cria o arquivo de teste com o código padrão:

```
Python

import unittest

class Test_test1(unittest.TestCase):
    def test_A(self):
        self.fail("Not implemented")

if __name__ == '__main__':
    unittest.main()
```

Esse código importa o módulo padrão `unittest` e deriva uma classe de teste do método `unittest.TestCase`. Ao executar o script diretamente, esse código também

invoca a função `unittest.main()`.

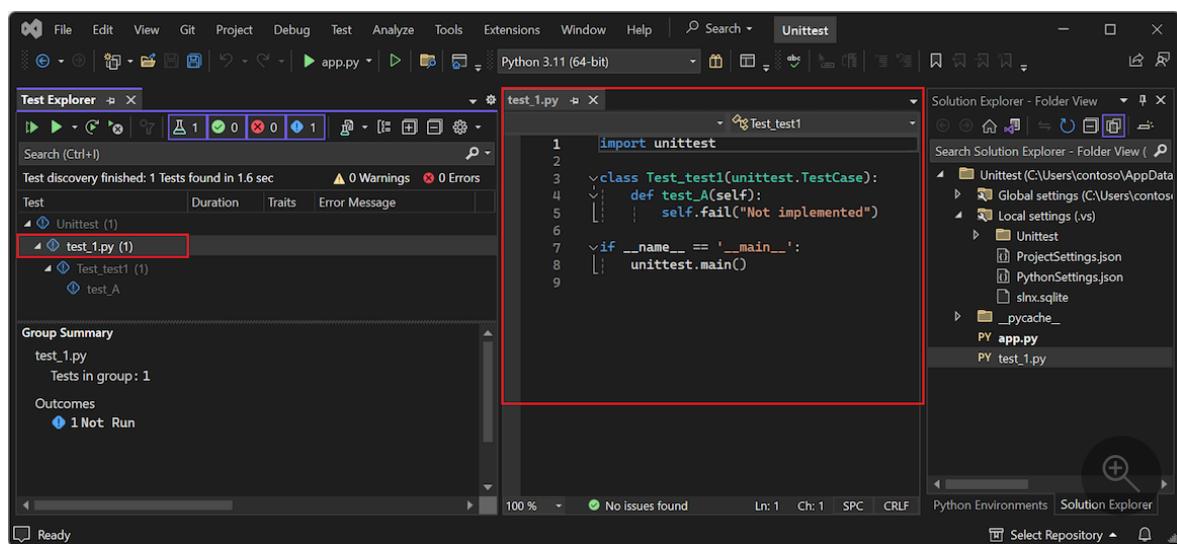
Ao adicionar novos arquivos de teste, o Visual Studio os disponibiliza no [Gerenciador de Testes](#).

Exibição de testes com o Gerenciador de Testes

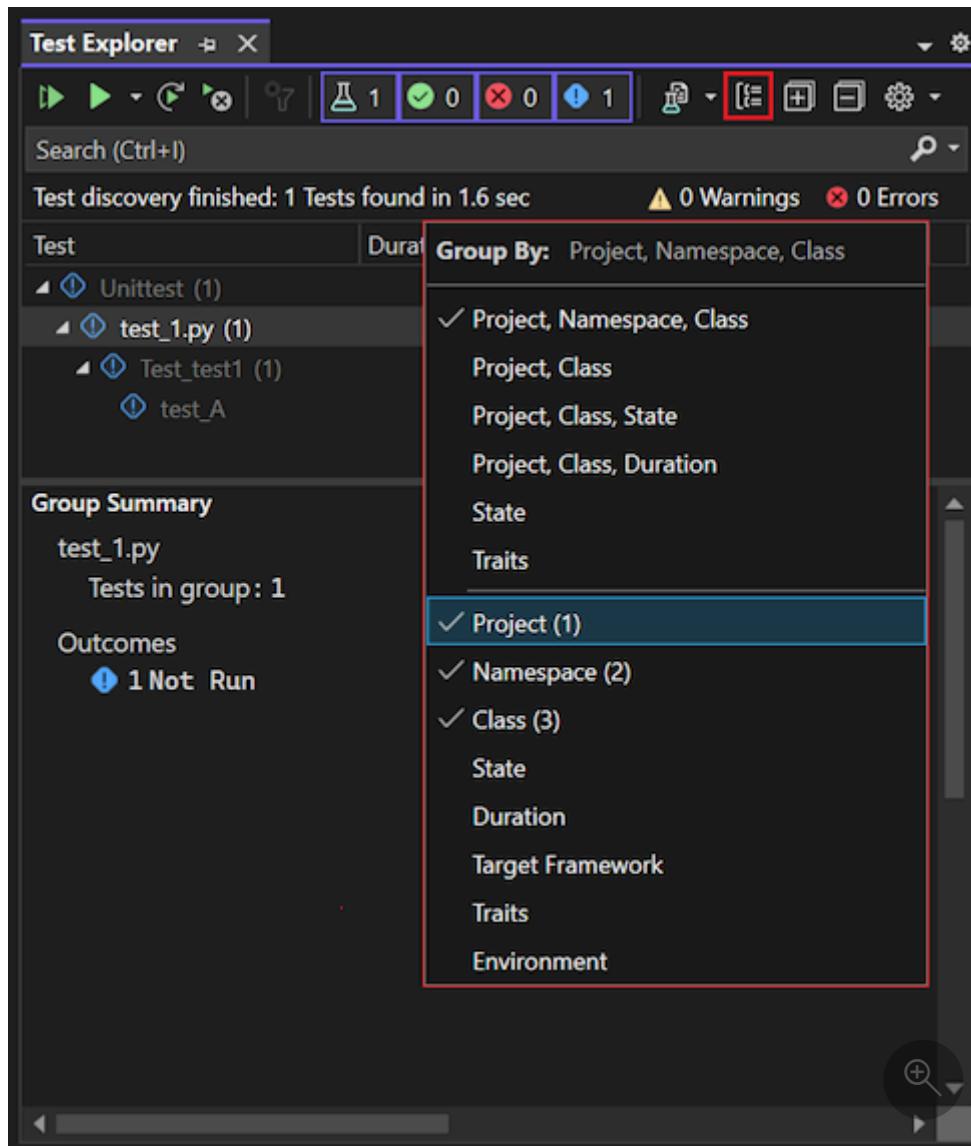
Após configurar a estrutura de teste e os arquivos de teste, o Visual Studio pesquisa os testes e os exibe no [Gerenciador de Testes](#).

Confira abaixo algumas maneiras para se trabalhar com o [Gerenciador de Testes](#):

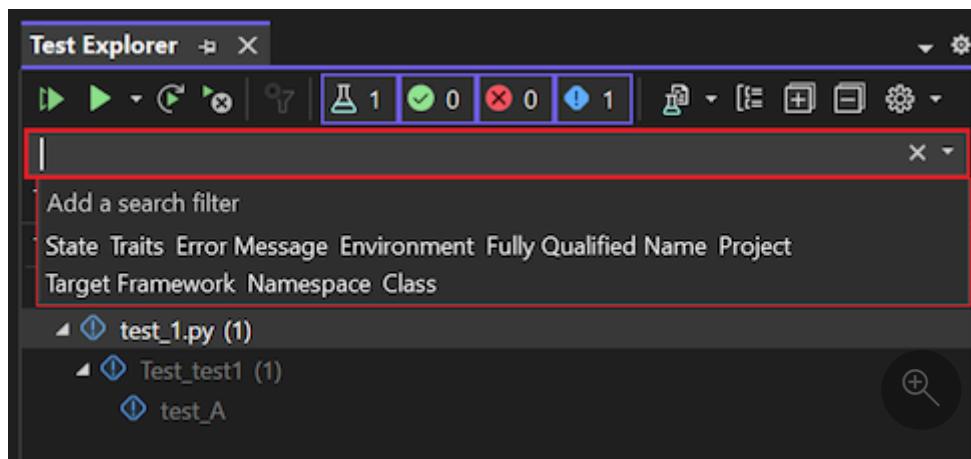
- Abra a janela **Gerenciador de Testes** ao selecionar **Testes>Gerenciador de Testes**.
- Quando a janela **Gerenciador de Testes** estiver aberta, use o atalho de teclado **CTRL+R, A** para acionar a detecção de testes.
- Realize um clique duplo em um teste no **Gerenciador de Testes** para abrir o arquivo de origem correspondente no editor:



- Organize a exibição dos testes ao usar a opção **Agrupar por** na barra de ferramentas:



- Filtre os testes por nome ao inserir texto no campo Pesquisar:



- Execute testes e realize a exibição do status da execução de teste, conforme descrito na próxima seção.

Para obter mais informações sobre o módulo `unittest` e realizar a gravação de testes, confira a [documentação do Python](#).

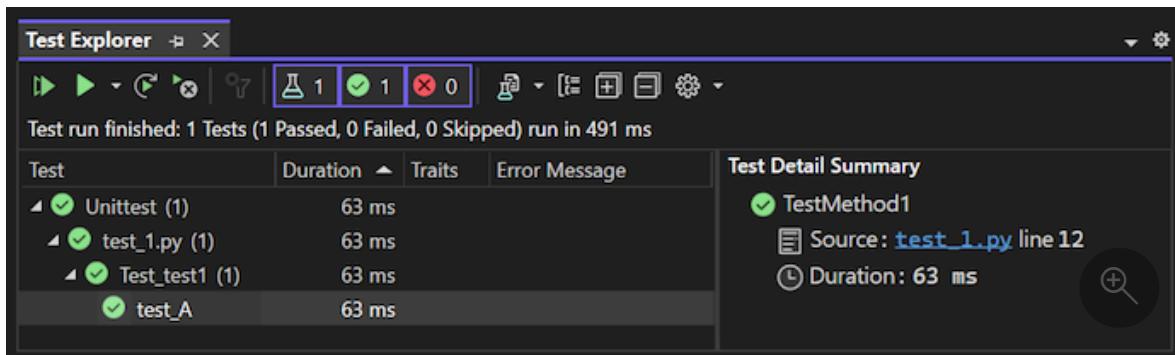
Executar testes com o Gerenciador de Testes

No Gerenciador de Testes, é possível realizar a execução de testes de diversas maneiras:

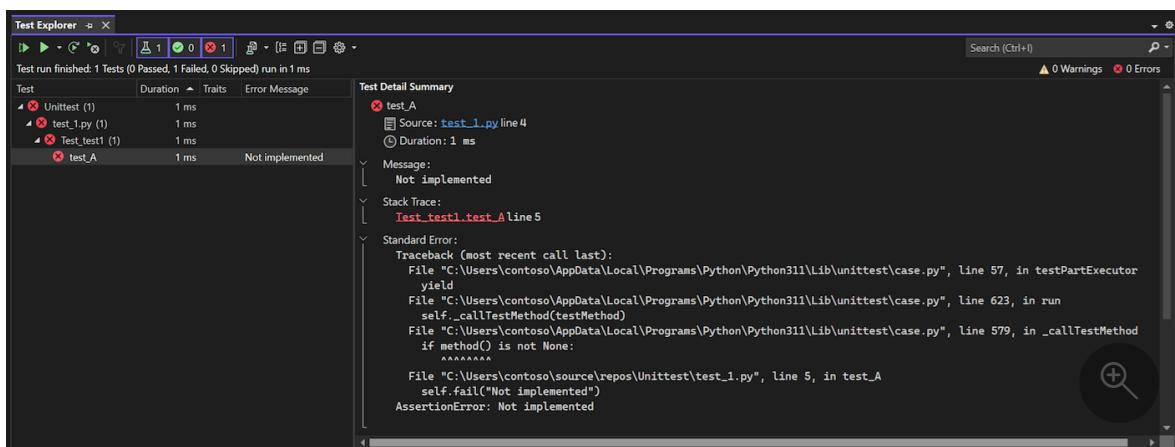
- Selecione **Executar Todos** (os testes na exibição) para executar todos os testes mostrados na exibição atual com base nas configurações do filtro.
- Use os comandos do menu **Executar** para realizar a execução de testes com falha, aprovados ou para realizar uma execução sem ser em grupo.
- Selecione um ou mais testes e, em seguida, clique com o botão direito do mouse e selecione a opção **Executar Testes Selecionados**.

O Visual Studio executará os testes em segundo plano. O **Gerenciador de Testes** atualiza o status de cada teste à medida que ele é concluído:

- Os testes **aprovados** apresentam um tique verde e o tempo para a conclusão da execução de teste:



- Os testes **com falha** apresentam um X vermelho com um link de **Saída** que mostra a saída do console e a saída `unittest` da execução de teste:



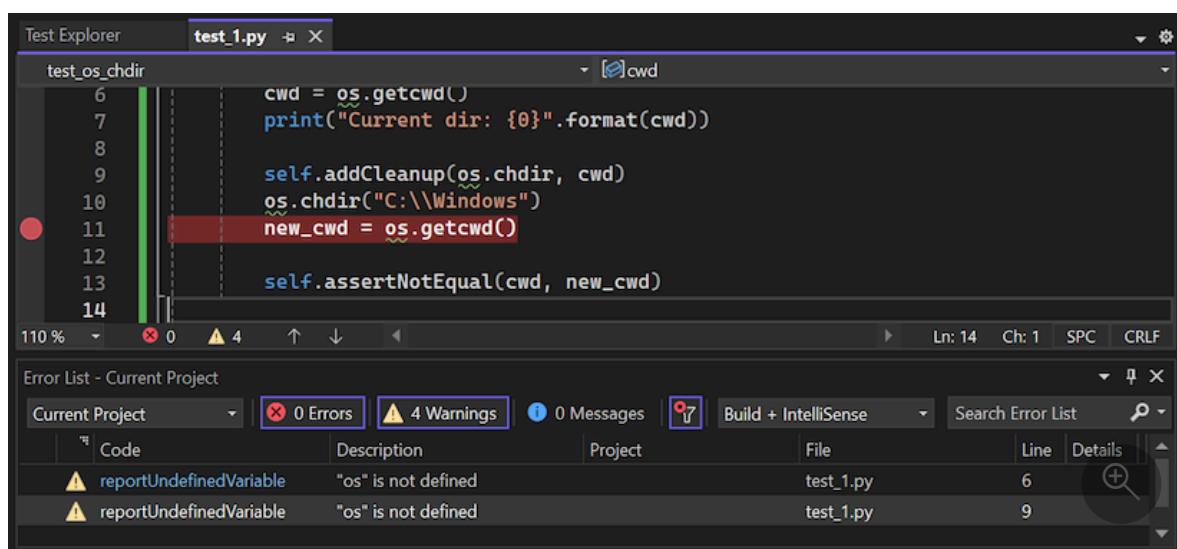
Verificação de testes com o Depurador

Os testes de unidades são segmentos de código suscetíveis a bugs, como qualquer outro código, e às vezes requerem execução em um depurador. No **Depurador** do

Visual Studio, é possível definir pontos de interrupção, analisar variáveis e examinar o código. O Visual Studio também fornece ferramentas de diagnóstico para testes de unidade.

Realize uma revisão destes pontos sobre como verificar os testes com o **Depurador do Visual Studio**:

- Por padrão, a depuração de teste usa o depurador **debugpy** para a versão 16.5 e para versões posteriores do Visual Studio 2019. Algumas versões anteriores do Visual Studio usam o depurador **ptvsd 4**. Se você estiver usando uma versão anterior do Visual Studio e preferir o depurador **ptvsd 3**, selecione a opção **Usar o Depurador Herdado em Ferramentas>Opções>Python>Depuração**.
- Para iniciar a depuração, defina um ponto de interrupção inicial no código, clique com o botão direito do mouse no teste (ou em uma seleção) no **Gerenciador de Testes** e escolha **Depurar Testes Selecionados**. O Visual Studio inicia o depurador do Python como faria com o código do aplicativo.



- Se preferir, é possível usar **Analizar Cobertura de Código para Testes Selecionados**. Para obter mais informações, confira [Usar a cobertura de código para determinar quanto do código está sendo testado](#).

Conteúdo relacionado

- [Ferramentas e tarefas de teste de unidade](#)
- [Introdução ao teste de unidade](#)

Comentários

Esta página foi útil?

 Yes

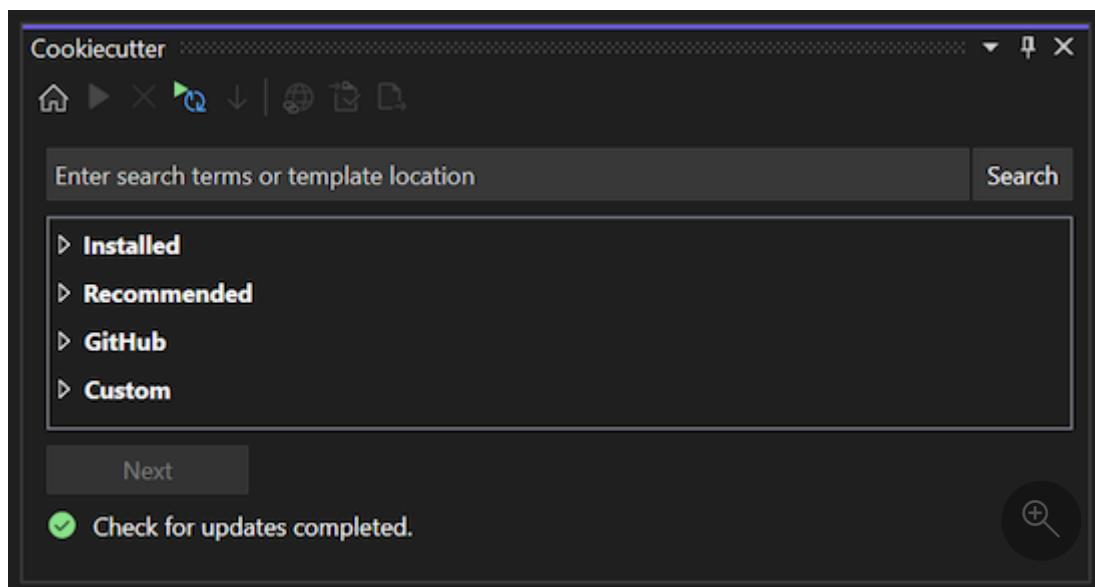
 No

Usar a extensão Cookiecutter

Artigo • 18/04/2024

O [Cookiecutter](#) fornece uma interface gráfica do usuário para descobrir modelos e opções de modelo de entrada e criar projetos e arquivos. O Visual Studio 2017 e posterior inclui a extensão Cookiecutter. Ela pode ser instalada separadamente em versões anteriores do Visual Studio.

No Visual Studio, a extensão Cookiecutter está disponível em **View>Cookiecutter Explorer**:



Pré-requisitos

- Visual Studio. Para instalar o produto, siga as etapas em [Instalar o Visual Studio](#).
- Python 3.3 ou posterior (32 ou 64 bits) ou o Anaconda 3 4.2 ou posterior (32 ou 64 bits).
 - Se um interpretador do Python adequado não estiver disponível, o Visual Studio exibirá um aviso.
 - Se você instalar um interpretador do Python enquanto o Visual Studio estiver em execução, selecione a opção **Início** na barra de ferramentas do **Cookiecutter Explorer** para detectar o interpretador recém-instalado. Para obter mais informações, consulte [Criar e gerenciar ambientes do Python no Visual Studio](#).

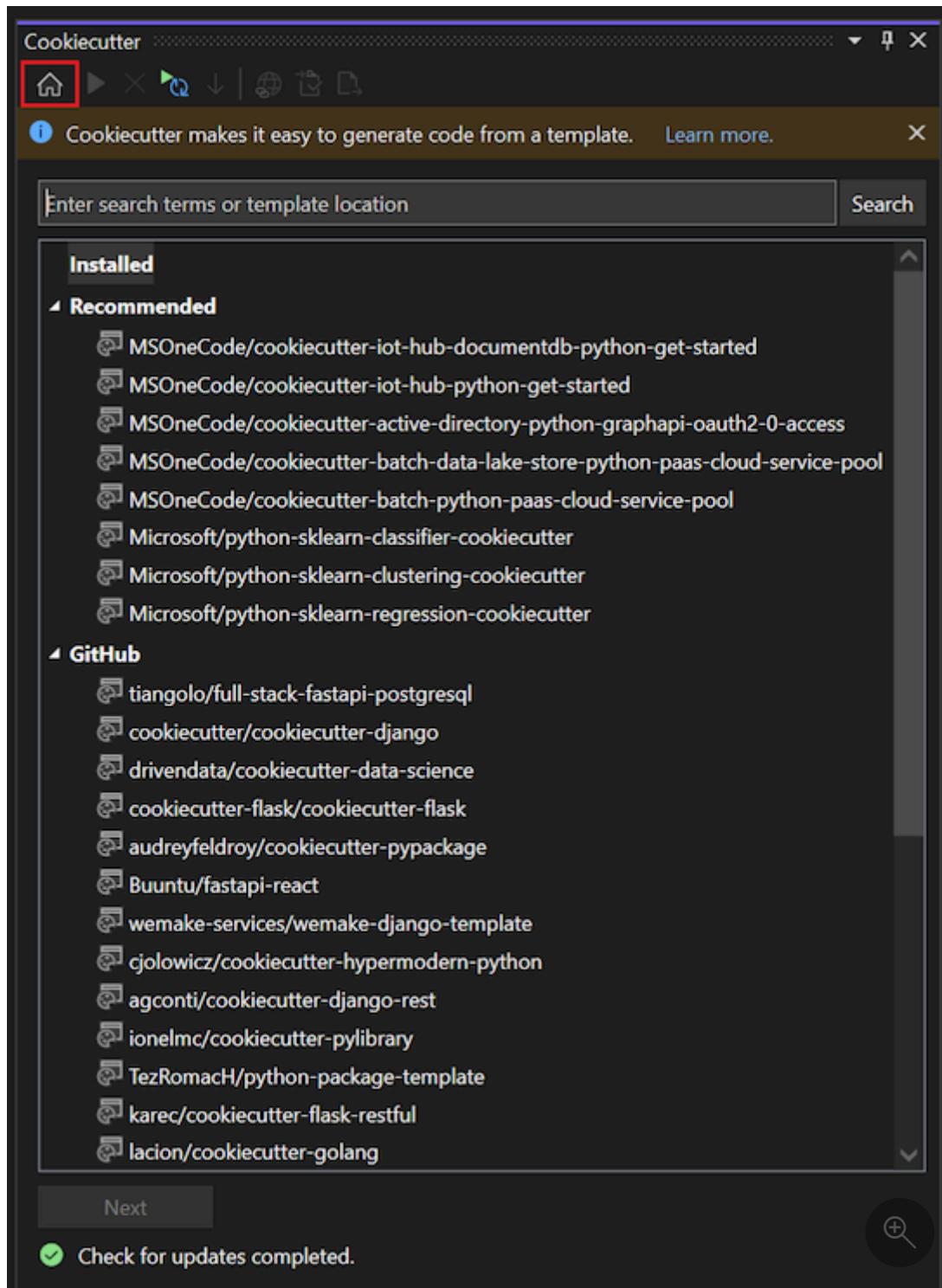
Trabalhar com o Cookiecutter Explorer

No **Cookiecuter Explorer**, você pode procurar e selecionar modelos, clonar modelos para seu computador local, definir opções de modelo e criar código a partir de modelos.

Procurar modelos

Você pode procurar modelos no **Cookiecuter Explorer** para ver o que já está instalado e o que está disponível.

1. No **Cookiecuter Explorer**, selecione a opção **Início** na barra de ferramentas para exibir os modelos disponíveis.



A home page exibe uma lista de modelos a serem escolhidos, organizados em quatro possíveis grupos:

 Expandir a tabela

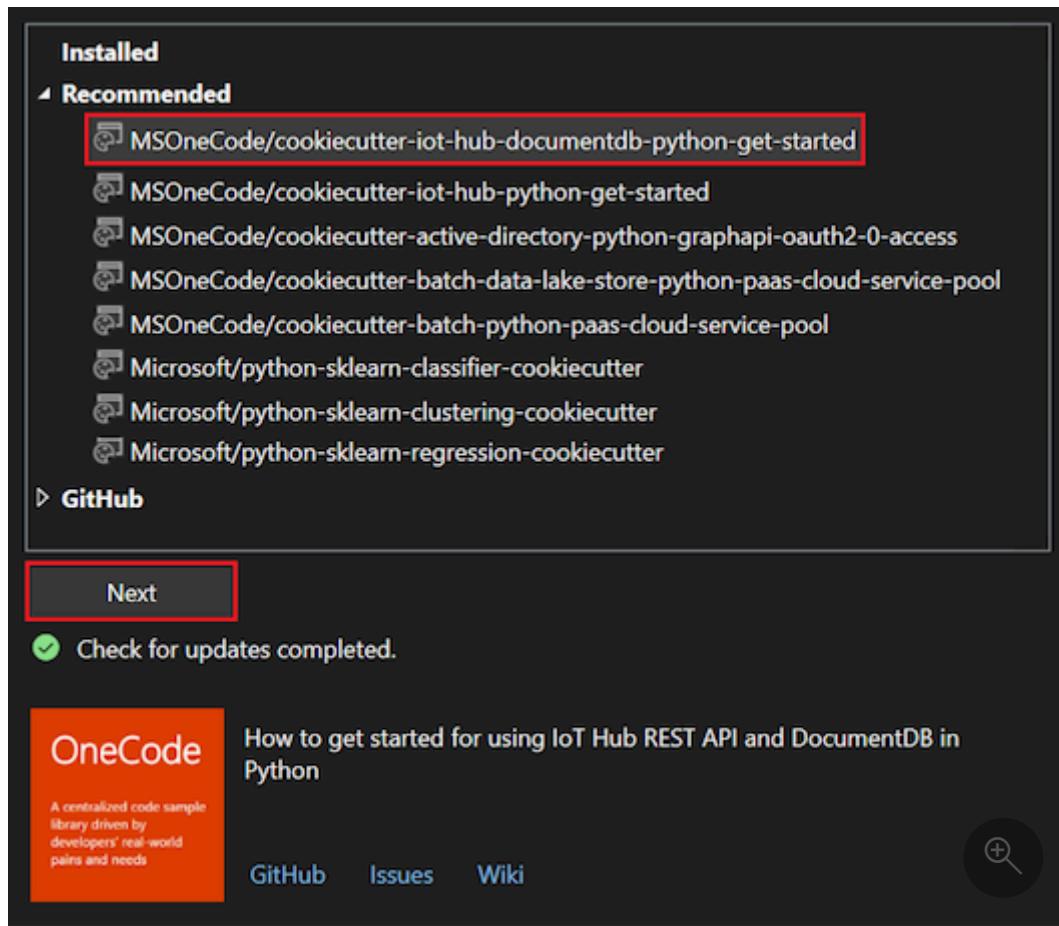
| Grupo | Descrição | Observações |
|---------------|--|---|
| Instalado | Modelos instalados no computador local. Quando um modelo online é usado, seu repositório é clonado automaticamente em uma subpasta de <code>~/.cookiecutters</code> . | Você pode remover um modelo instalado do sistema selecionando Excluir na barra de ferramentas do Cookiecutter Explorer . |
| Recomendado | Modelos carregados do feed recomendado. A Microsoft organiza o feed padrão. | Você pode personalizar o feed seguindo as etapas em Definir opções do Cookiecutter . |
| GitHub | Resultados da pesquisa do GitHub para a palavra-chave "cookiecutter". A lista de repositórios git é retornada em formato paginado. | Quando a lista de resultados excede a exibição atual, você pode selecionar a opção Carregar Mais para mostrar o próximo conjunto de resultados paginados na lista. |
| Personalizado | Quaisquer modelos personalizados definidos por meio do Cookiecutter Explorer . Quando um local de modelo personalizado é inserido na caixa de pesquisa do Cookiecutter Explorer , o local aparece nesse grupo. | Você pode definir um modelo personalizado inserindo o caminho completo para o repositório git ou o caminho completo para uma pasta no disco local. |

2. Para mostrar ou ocultar a lista de modelos disponíveis para uma categoria específica, selecione a seta ao lado da categoria.

Clonar modelos

Você pode trabalhar com modelos disponíveis no **Cookiecutter Explorer** para fazer cópias locais para trabalhar.

1. No **Cookiecutter Explorer**, selecione um modelo. As informações sobre o modelo selecionado são exibidas na parte inferior da home page do **Cookiecutter Explorer**



O resumo do modelo inclui links para obter mais informações sobre o modelo. Você pode ir para a página do repositório do **GitHub** referente ao modelo, visualizar o **Wiki** do modelo ou encontrar os **Problemas** relatados.

2. Para clonar o modelo selecionado, selecione **Avançar**. O Cookiecutter faz uma cópia local do modelo.

O comportamento de clonagem depende do tipo de modelo selecionado:

[+] Expandir a tabela

| Tipo do modelo | Comportamento |
|------------------------|--|
| Instalado | Se o modelo selecionado foi instalado em uma sessão anterior do Visual Studio, ele é excluído automaticamente, e a versão mais recente é instalada e clonada no computador local. |
| Recomendado | O modelo selecionado é clonado e instalado no computador local. |
| GitHub | O modelo selecionado é clonado e instalado no computador local. |
| Pesquisa personalizada | <ul style="list-style-type: none">- URL: se você inserir um URL personalizado para um repositório git na caixa de pesquisa do Cookiecutter Explorer e, em seguida, selecionar o modelo, o modelo selecionado será clonado e instalado no computador local.- Caminho da pasta: se você inserir um caminho de pasta personalizado na |

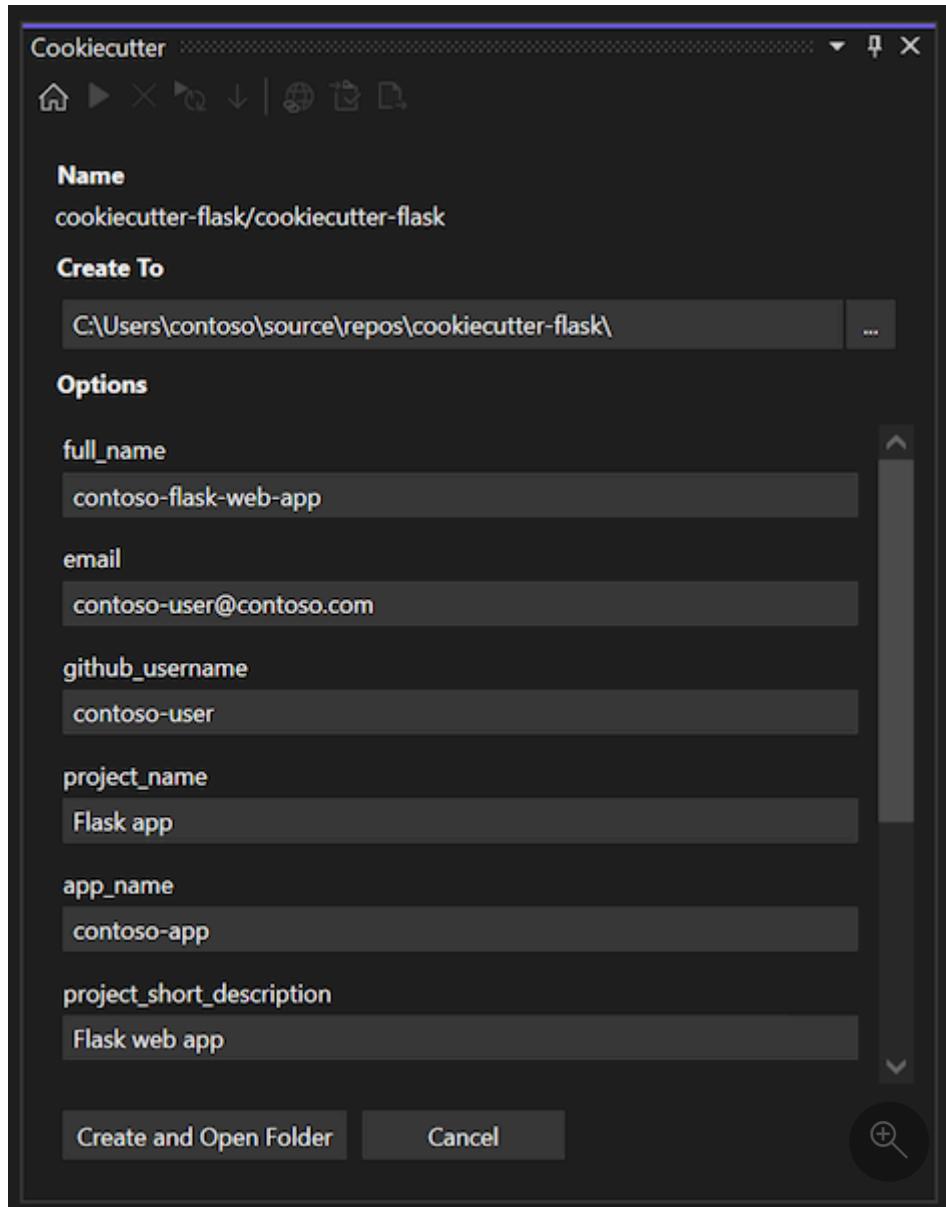
| Tipo do modelo | Comportamento |
|-----------------------|--|
| | caixa de pesquisa e selecionar o modelo, o Visual Studio carregará esse modelo sem clonagem. |

ⓘ Importante

Os modelos do Cookiecutter são clonados em uma única pasta `~/.cookiecutters`. Cada subpasta é nomeada de acordo com o nome do repositório Git, que não inclui o nome de usuário do GitHub. Poderão surgir conflitos se você clonar modelos diferentes com o mesmo nome que são de autores diferentes. Nesse caso, o Cookiecutter impede que você substitua o modelo existente por um modelo diferente com o mesmo nome. Para instalar o outro modelo, é necessário primeiro excluir existente.

Configurar opções de modelo

Depois de instalar e clonar um modelo localmente, o Cookiecutter exibe a página **Opções**. Nessa página, você pode especificar configurações, como o local do caminho da pasta para os arquivos gerados:



Cada modelo do Cookiecutter define seu próprio conjunto de opções. Quando um valor padrão está disponível para uma configuração, a página **Opções** mostra o texto sugerido no campo correspondente. Um valor padrão pode ser um snippet de código, geralmente, quando ele é um valor dinâmico que usa outras opções.

Para este exemplo, o nome do modelo é definido como **cookiecutter-flask/cookiecutter-flask**. Quando um valor de configuração pode ser alterado, o texto do campo fica disponível para edição.

1. No campo **Criar para**, insira o local do caminho da pasta para todos os arquivos gerados pelo Cookiecutter.
2. Em seguida, defina outras opções desejadas para o modelo, como:
 - **full_name**: o nome completo a ser aplicado ao modelo.
 - **email**: o endereço de e-mail do autor do modelo.
 - **github_username**: o alias do GitHub do autor do modelo.

- `python_version`: a versão Python de destino para aplicativos Web criados a partir do modelo.

Definir padrões com um arquivo de configuração

Você pode personalizar os valores padrão de opções específicas com um arquivo de configuração do usuário. Quando a extensão Cookiecutter detecta um arquivo de configuração do usuário, ela substitui os valores padrão do modelo pelos valores do arquivo de configuração. Para obter mais informações sobre esse comportamento, consulte a seção [Configuração do usuário](#) da documentação do Cookiecutter.

Desativar tarefas especificadas

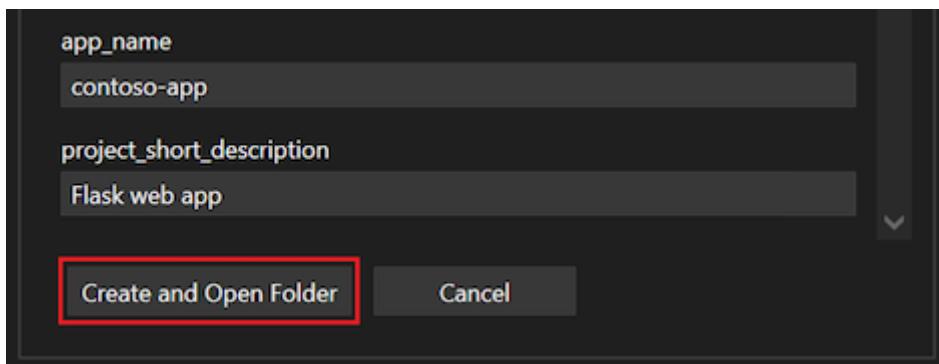
Alguns modelos identificam tarefas específicas do Visual Studio a serem executadas após a geração de código. As tarefas comuns incluem abrir um navegador da Web, abrir arquivos no editor e instalar dependências. Quando um modelo identifica tarefas específicas, a configuração **Executar tarefas adicionais ao concluir** é adicionada à lista de opções. Você pode definir essa configuração para desativar as tarefas especificadas do Visual Studio.

Criar código com base em modelos

Depois de definir suas opções de modelo, você estará pronto para usar o Cookiecutter para criar os arquivos de projeto e gerar o código.

A caixa de diálogo exibe um botão após a lista de opções. O texto do botão depende do modelo. Você pode ver **Criar e abrir pasta**, **Adicionar à solução** e assim por diante.

1. Na página **Opções** selecione o botão que segue a lista de opções, como **Criar e abrir pasta** ou **Adicionar à solução**.



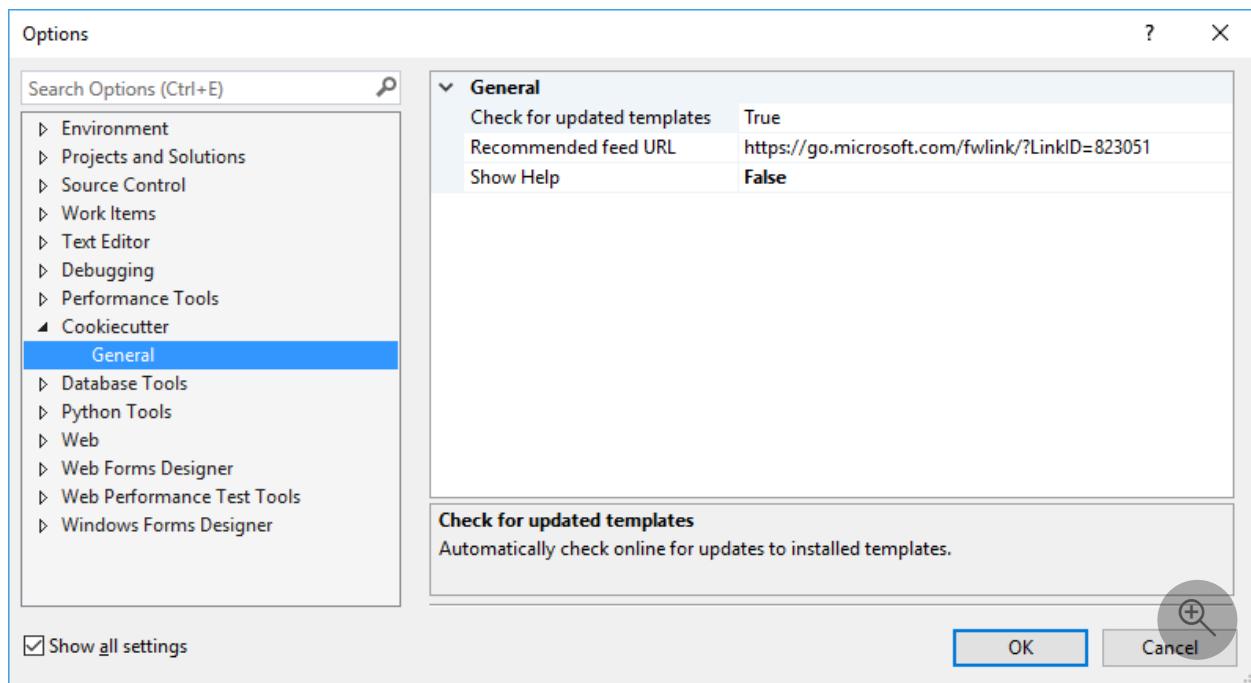
O Cookiecutter gera o código. Se a pasta de saída não estiver vazia, um aviso será exibido.

- Caso você já conheça a saída do modelo e não se incomode em substituir arquivos, selecione **OK** para ignorar o aviso.
- Caso contrário, selecione **Cancelar**, especifique uma pasta vazia e, em seguida, copie manualmente os arquivos criados para a pasta de saída não vazia.

2. Depois que o Cookiecutter criar com êxito os arquivos, o Visual Studio abrirá os arquivos de projeto de modelo no **Gerenciador de Soluções**.

Definir as opções do Cookiecutter

As opções do Cookiecutter estão disponíveis por meio de **Ferramenta > Opções > Cookiecutter**:



Expandir a tabela

| Opção | Descrição |
|-------------------------------------|--|
| Verificar se há modelos atualizados | Controla se o Cookiecutter verifica automaticamente online se há atualizações para os modelos instalados. |
| URL do feed recomendado | A localização do arquivo de feed recomendado dos modelos. O local pode ser um URL ou um caminho para um arquivo local. Deixe a URL vazia para usar o feed padrão coletado pela Microsoft. O feed fornece uma lista simples de localizações de modelos, separadas por novas linhas. Para solicitar alterações ao feed coletado, faça uma solicitação pull na fonte, no GitHub . |

| Opção | Descrição |
|---------------|---|
| Mostrar Ajuda | Controla a visibilidade da barra de informações de ajuda na parte superior da janela do Cookiecutter. |

Otimizar modelos do Cookiecutter para o Visual Studio

A extensão Cookiecutter para o Visual Studio dá suporte aos modelos criados para o Cookiecutter v1.4. Para obter mais informações sobre como criar modelos do Cookiecutter, consulte a [documentação do Cookiecutter](#).

A renderização padrão de uma variável de modelo depende do tipo de dados (cadeia de caracteres ou lista):

- **String:** o tipo de dados String usa um rótulo para o nome da variável, uma caixa de texto para inserir o valor e uma marca d'água que mostra o valor padrão. Uma dica de ferramenta na caixa de texto mostra o valor padrão.
- **List:** o tipo de dados List usa um rótulo para o nome da variável e uma caixa de combinação para selecionar um valor. Uma dica de ferramenta na caixa de combinação mostra o valor padrão.

Você pode melhorar a renderização especificando outros metadados adicionais no arquivo `cookiecutter.json` que são específicos ao Visual Studio (e ignorados pela CLI do Cookiecutter). Todas as propriedades são opcionais:

[+] [Expandir a tabela](#)

| Propriedade | Descrição |
|--------------------------|--|
| <code>label</code> | Especifica o texto a ser exibido acima do editor para a variável, em vez do nome da variável. |
| <code>description</code> | Especifica a dica de ferramenta a ser exibida no controle de edição, em vez do valor padrão dessa variável. |
| <code>url</code> | Altera o rótulo para um hiperlink, com uma dica de ferramenta que mostra a URL. Clicar no hiperlink abre o navegador padrão do usuário nessa URL. |
| <code>selector</code> | Permite a personalização do editor para uma variável. Atualmente, há suporte para os seguintes seletores: <ul style="list-style-type: none"> - <code>string</code>: caixa de texto padrão, o padrão para cadeias de caracteres. - <code>list</code>: caixa de combinação padrão, o padrão para listas. - <code>yesno</code>: caixa de combinação para escolher entre <code>y</code> e <code>n</code>, para cadeias de caracteres. |

| Propriedade | Descrição |
|-------------------------------|--|
| - <code>odbcConnection</code> | Caixa de texto com um botão de reticências (...) que abre uma caixa de diálogo de conexão de banco de dados. |

O exemplo a seguir mostra como definir propriedades de renderização:

```
JSON

{
    "site_name": "web-app",
    "python_version": ["3.5.2"],
    "use_azure": "y",

    "_visual_studio": {
        "site_name": {
            "label": "Site name",
            "description": "E.g. <site-name>.azurewebsites.net (can only contain alphanumeric characters and `-'`)"
        },
        "python_version": {
            "label": "Python version",
            "description": "The version of Python to run the site on"
        },
        "use_azure" : {
            "label": "Use Azure",
            "description": "Include Azure deployment files",
            "selector": "yesno",
            "url": "https://azure.microsoft.com"
        }
    }
}
```

Executar tarefas do Visual Studio

O Cookiecutter tem um recurso chamado **Pós-gerar Ganchos**, que permite executar código arbitrário do Python após a geração dos arquivos. Embora o recurso seja flexível, ele não permite fácil acesso ao Visual Studio.

Você pode usar esse recurso para abrir um arquivo no editor do Visual Studio ou seu navegador da Web. Você também pode disparar a interface do usuário do Visual Studio que solicita que o usuário crie um ambiente virtual e instale os requisitos do pacote.

Para permitir esses cenários, o Visual Studio procura metadados estendidos no arquivo `cookiecutter.json`. Ele procura os comandos a serem executados depois que o usuário abre os arquivos gerados no **Gerenciador de Soluções** ou depois que os arquivos são adicionados a um projeto existente. (Novamente, o usuário pode recusar a execução das tarefas desmarcando a opção de modelo **Executar tarefas adicionais após a conclusão**.)

O exemplo a seguir mostra como definir metadados estendidos no arquivo `cookiecutter.json`:

JSON

```
"_visual_studio_post_cmds": [
    {
        "name": "File.OpenFile",
        "args": "{{cookiecutter._output_folder_path}}\\readme.txt"
    },
    {
        "name": "Cookiecutter.ExternalWebBrowser",
        "args": "https://learn.microsoft.com"
    },
    {
        "name": "Python.InstallProjectRequirements",
        "args": "{{cookiecutter._output_folder_path}}\\dev-requirements.txt"
    }
]
```

Especifique os comandos pelo nome e use o nome não localizado (em inglês) para funcionar em instalações localizadas do Visual Studio. É possível testar e descobrir os nomes de comando na janela **Comando** do Visual Studio.

Se você quiser passar um único argumento, especifique o argumento como uma cadeia de caracteres, conforme mostrado para os metadados `name` no exemplo anterior.

Se não precisar passar um argumento, deixe o valor como uma cadeia de caracteres vazia ou omita-o do arquivo JSON:

JSON

```
"_visual_studio_post_cmds": [
    {
        "name": "View.WebBrowser"
    }
]
```

Para vários argumentos, use uma matriz. Para opções, divida a opção e seu valor em argumentos separados e use a delimitação correta, conforme mostrado neste exemplo:

JSON

```
"_visual_studio_post_cmds": [
    {
        "name": "File.OpenFile",
        "args": [
            "{{cookiecutter._output_folder_path}}\\read_me.txt",
            "-nologo"
        ]
    }
]
```

```
        "/e:",
        "Source Code (text) Editor"
    ]
}
```

Os argumentos podem se referir a outras variáveis do Cookiecutter. No exemplo anterior, a variável `_output_folder_path` interna é usada para formar um caminho absoluto para os arquivos gerados.

O comando `Python.InstallProjectRequirements` apenas funciona ao adicionar arquivos a um projeto existente. Essa limitação existe porque o comando é processado pelo projeto Python no **Gerenciador de Soluções** e não há nenhum projeto para receber a mensagem no **Gerenciador de Soluções - Exibição de Pasta**.

Solucionar problemas de modelo

Veja as seções a seguir para obter dicas sobre como solucionar problemas no ambiente e código Python ao trabalhar com o Cookiecutter.

Erro ao carregar o modelo

Alguns modelos podem usar tipos de dado inválidos no arquivo `cookiecutter.json`, como o booliano. Você pode reportar essas instâncias para o autor do modelo selecionando o link **Problemas** no painel de informações do modelo.

Script de gancho com falha

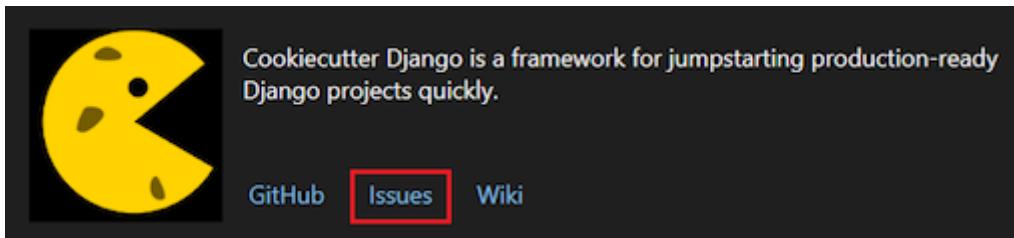
Alguns modelos podem usar scripts pós-geração que não são compatíveis com a interface do usuário do Cookiecutter. Por exemplo, os scripts que consultam a entrada do usuário podem falhar devido a uma falta de console de terminal.

Script de gancho sem suporte no Windows

Se o arquivo pós-script for `.sh`, ele poderá não ser associado a um aplicativo no computador Windows. Você poderá ver uma caixa de diálogo do Windows pedindo para você encontrar um aplicativo compatível na Windows Store.

Modelos com problemas conhecidos

Você pode descobrir se um modelo tem problemas conhecidos usando o link Problemas no resumo do modelo no **Cookiecutter Explorer**:



O link abre a página de problemas do GitHub para o modelo:

Conteúdo relacionado

- Referência de modelos de item do Python
- Cookiecutter: melhores modelos de projeto ↗

Comentários

Esta página foi útil?

Yes

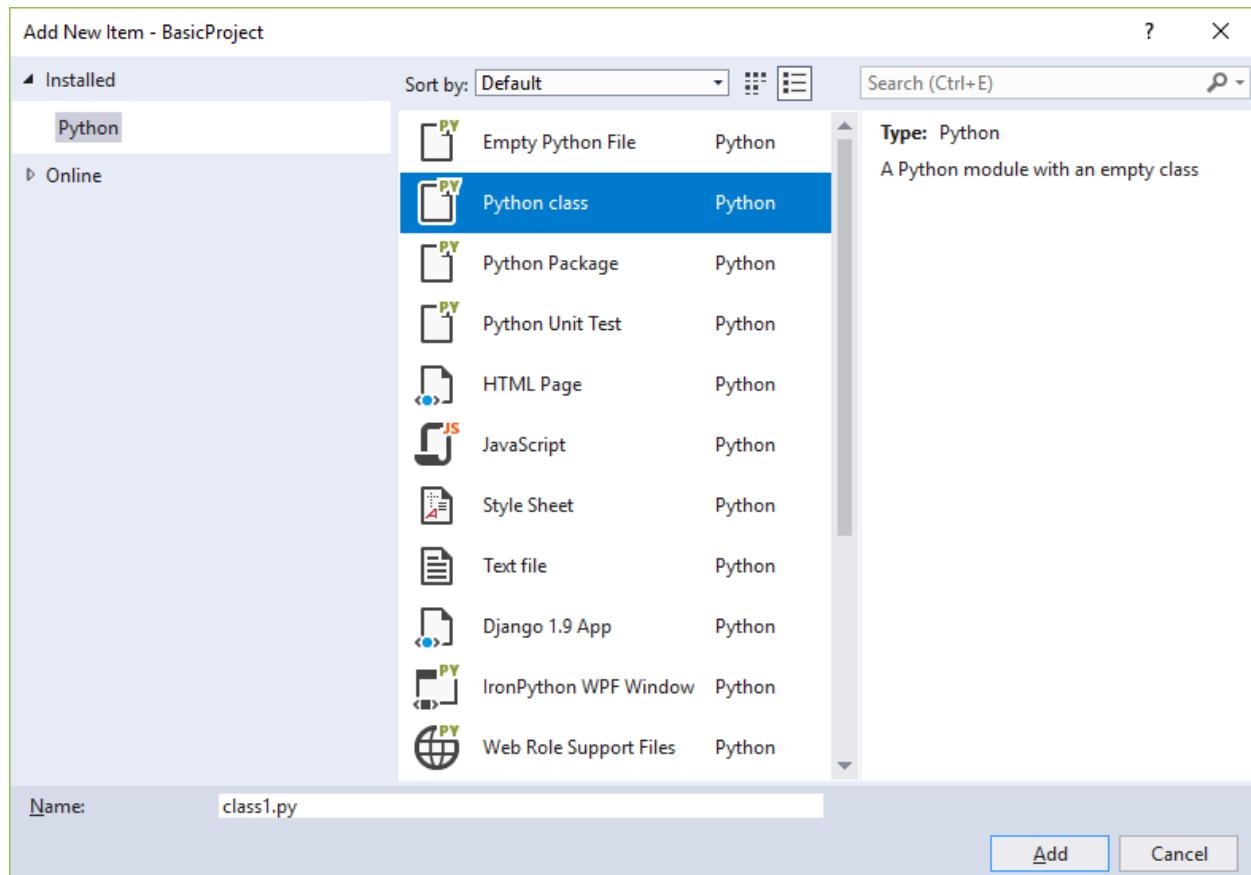
No

Modelos de item do Python

Artigo • 19/06/2023

Aplica-se a: Visual Studio Visual Studio para Mac Visual Studio Code

Os modelos de item estão disponíveis em projetos do Python através do comando de menu **Projeto>Adicionar Novo Item** ou do comando **Adicionar>Novo Item** no menu de contexto no **Gerenciador de Soluções**.



Se você usar o nome fornecido para o item, o modelo geralmente criará um ou mais arquivos e pastas dentro da pasta que está atualmente marcada no projeto (ao clicar duas vezes com o botão direito do mouse em uma pasta para exibir o menu de contexto, essa pasta será automaticamente marcada). Se você adicionar um item, ele será incluído no projeto do Visual Studio e será exibido no **Gerenciador de Soluções**.

A tabela a seguir explica brevemente o efeito de cada modelo de item em um projeto do Python:

| Modelo | O que o modelo cria |
|-------------------------|--------------------------------------|
| Arquivo vazio do Python | Um arquivo vazio com a extensão .py. |

| Modelo | O que o modelo cria |
|--|--|
| Classe Python | Um arquivo <code>.py</code> que contém uma única definição de classe vazia do Python. |
| Pacote do Python | Uma pasta que contém um arquivo <code>__init__.py</code> . |
| Teste de Unidade do Python | Um arquivo <code>.py</code> com um único teste de unidade baseado na estrutura <code>unittest</code> , juntamente com uma chamada a <code>unittest.main()</code> para executar os testes no arquivo. |
| Página HTML | Um arquivo <code>.html</code> com uma estrutura de página simples que consiste em um <code><head></code> e um elemento <code><body></code> . |
| JavaScript | Um arquivo <code>.js</code> vazio. |
| Folha de Estilos | Um arquivo <code>.css</code> que contém um estilo vazio para <code>body</code> . |
| Arquivo de texto | Um arquivo <code>.txt</code> vazio. |
| Aplicativo Django 1.9 Aplicativo Django 1.4 | Uma pasta com o nome do aplicativo que contém os arquivos principais de um aplicativo do Django conforme explicado em Aprender Django no Visual Studio, Etapa 2 de 2 para o Django 1.9. No caso do Django 1.4, a pasta <code>migrations</code> , o arquivo <code>admin.py</code> e o arquivo <code>apps.py</code> não estão incluídos. |
| Arquivos de Suporte de Função da Web | Uma pasta <code>bin</code> na raiz do projeto (independentemente da pasta escolhida no projeto). A pasta contém um script de implantação padrão e um arquivo <code>web.config</code> para funções da web do Serviço de Nuvem do Azure. O modelo também inclui um arquivo <code>readme.html</code> que explica os detalhes. |
| Arquivos de suporte à função de trabalho | Uma pasta <code>bin</code> na raiz do projeto (independentemente da pasta escolhida no projeto). A pasta contém o script de implantação e lançamento padrão, além de um arquivo <code>web.config</code> , para funções de trabalho do Serviço de Nuvem do Azure. O modelo também inclui um arquivo <code>readme.html</code> que explica os detalhes. |
| web.config do Azure (FastCGI) | Um arquivo <code>web.config</code> que contém entradas para aplicativos que usam um objeto WSGI para tratar das conexões de entrada. Normalmente, esse arquivo é implantado na raiz de um servidor Web que executa o IIS. Para saber mais, confira Configurar um aplicativo para IIS . |
| web.config do Azure (HttpPlatformHandler) | Um arquivo <code>web.config</code> que contém entradas para aplicativos que escutam conexões de entrada com um soquete. Normalmente, esse arquivo é implantado na raiz de um servidor Web que executa o IIS, como o Serviço de Aplicativo do Azure. Para saber mais, confira Configurar um aplicativo para IIS . |

| Modelo | O que o modelo cria |
|--|--|
| Arquivos estáticos web.config do Azure | Um arquivo <i>web.config</i> normalmente adicionado a uma pasta <i>static</i> (ou outra pasta que contém itens estáticos) para desabilitar o processamento do Python para essa pasta. Esse arquivo de configuração funciona em conjunto com um dos arquivos de configuração FastCGI ou HttpPlatformHandler acima. Para saber mais, confira Configurar um aplicativo para IIS . |
| Depuração remota de web.config do Azure | Preterido (foi usado para depuração remota no Serviço de Aplicativo do Azure para Windows, que não é mais suportado). |

Confira também

- [Gerenciar projetos Python – Modelos de projetos](#)
- [Modelos de projeto Web do Python](#)

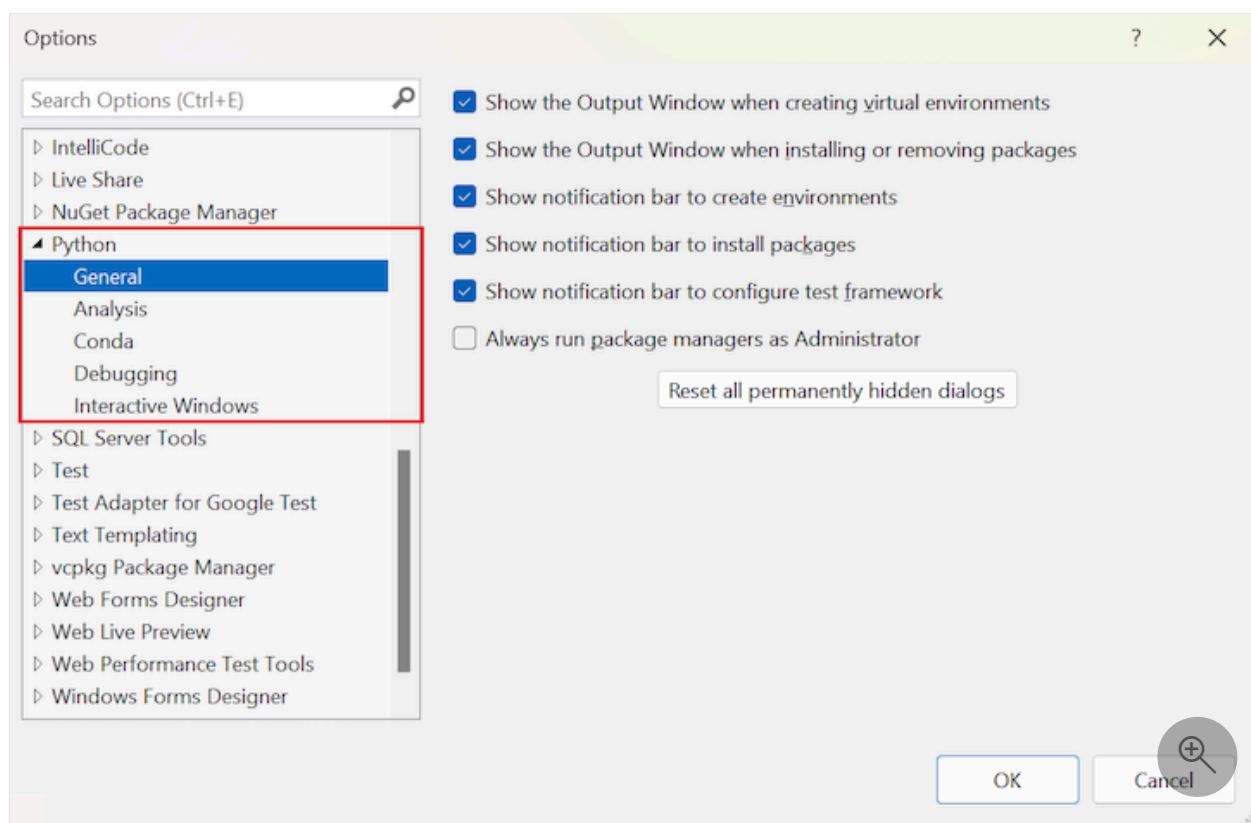
Opções para o Python no Visual Studio

Artigo • 18/04/2024

O Visual Studio fornece suporte na configuração do ambiente de desenvolvimento interativo (IDE) para desenvolvimento em Python. Você pode definir opções de acordo com suas preferências e para atender às necessidades específicas do ambiente de desenvolvimento. Este artigo descreve as opções disponíveis para layout e comportamento gerais, depuração, diagnóstico e recursos avançados da linguagem Python.

Local das opções para Python

As definições de configuração do Python estão disponíveis na barra de ferramentas do Visual Studio em **Ferramentas>Opções**. A caixa de diálogo **Opções** lista a maioria das configurações do Python na guia **Python**:



Você pode configurar preferências para depuração, análise de servidor de linguagem Pylance, ambientes do Conda, ambiente geral e **Janelas Interativas**.

A caixa de diálogo **Opções** lista outras configurações do Python em **Editor de Texto>Python**. Há opções para barras de rolagem, guias e formatação, além de configurações gerais e avançadas. Outras configurações estão disponíveis em **Ambiente>Fontes e Cores** para o grupo de configurações do **Editor de Texto**.

ⓘ Observação

A caixa de diálogo **Opções** pode incluir uma guia ou grupo **Experimental** para recursos em desenvolvimento que não são descritos neste artigo. Você encontra mais informações em postagens sobre a [engenharia Python no blog da Microsoft](#).

Opções específicas do Python

Em **Ferramentas>Opções>Python**, você pode definir opções específicas do Python para o ambiente geral, incluindo **Janelas Interativas**, ambientes do Conda, depuração e muito mais.

Opções gerais do Python

As seguintes opções estão disponíveis em **Ferramentas>Opções>Python>Geral**:

[+] Expandir a tabela

| Opção | Padrão | Descrição |
|--|---------|--|
| Mostrar a Janela de Saída ao criar ambientes virtuais | Ativado | Desmarque essa opção para impedir que a janela Saída seja exibida. |
| Mostrar a Janela de Saída ao instalar ou remover pacotes | Ativado | Desmarque essa opção para impedir que a janela Saída seja exibida. |
| Mostrar barra de notificações para criar ambientes | Ativado | Quando essa opção é configurada e o usuário abre um projeto que contém um arquivo <i>requirements.txt</i> ou <i>environment.yml</i> , o Visual Studio exibe uma barra de informações com sugestões para criar um ambiente virtual ou ambiente do Conda, respectivamente, em vez de usar o ambiente global padrão. |
| Mostrar barra de notificações para instalar pacotes | Ativado | Quando essa opção é configurada e o usuário abre um projeto que contém um arquivo <i>requirements.txt</i> que não usa o ambiente global padrão, o Visual Studio compara esses requisitos com pacotes instalados no ambiente atual. Se houver pacotes ausentes, o Visual Studio exibirá um prompt para instalar essas dependências. |

| Opção | Padrão | Descrição |
|--|------------|---|
| Mostrar barra de notificação para configurar a estrutura de teste | Ativado | Quando essa opção é configurada, se o Visual Studio detectar arquivos no projeto Python que possam conter testes, mas nenhuma estrutura de teste está habilitada, o Visual Studio solicita que você habilite pytest ou unittest. |
| Sempre executar gerenciadores de pacotes como administrador | Desativado | Sempre eleva <code>pip install</code> e operações semelhantes de gerenciador de pacote para todos os ambientes. Ao instalar pacotes, o Visual Studio solicita privilégios de administrador se o ambiente estiver localizado em uma área protegida do sistema de arquivos, como <code>c:\Program Files</code> . Nesse prompt, você pode optar por sempre elevar o comando de instalação apenas para esse ambiente específico. Para obter mais informações, consulte a guia Pacotes . |

Opções de ambiente do Conda

As seguintes opções estão disponíveis em **Ferramentas>Opções>Python>Conda**:

 [Expandir a tabela](#)

| Opção | Padrão | Descrição |
|---------------------------------------|---------|--|
| Caminho do executável do Conda | (blank) | Especifica um caminho exato para o arquivo executável <code>conda.exe</code> , em vez de contar com a instalação do Miniconda padrão incluído na carga de trabalho do Python. Se outro caminho for fornecido aqui, ele terá precedência sobre a instalação padrão e outros executáveis <code>conda.exe</code> especificados no registro. Essa configuração poderá ser alterada se você instalar manualmente uma versão mais recente do Anaconda ou do Miniconda ou se desejar usar uma distribuição de 32 bits em vez da distribuição padrão de 64 bits. |

Opções de depuração

As seguintes opções estão disponíveis em **Ferramentas>Opções>Python>Depuração**:

 [Expandir a tabela](#)

| Opção | Padrão | Descrição |
|--|---------|---|
| Perguntar antes de executar quando houver erros | Ativado | Quando essa opção é configurada, o Visual Studio solicita uma confirmação de que deseja executar o código que contém erros. Para desabilitar o aviso, desmarque essa opção. |

| Opção | Padrão | Descrição |
|--|----------------------|--|
| Aguardar pela entrada quando o processo for encerrado de forma anormal | Ativo (para os dois) | Um programa de Python iniciado no Visual Studio é executado em sua própria janela de console. Por padrão, a janela espera que você pressione uma tecla antes de fechá-la, independentemente de como o programa é encerrado. Para remover este prompt e fechar a janela automaticamente, desmarque uma ou ambas as opções. |
| Aguardar pela entrada quando o processo for encerrado normalmente | | |
| A saída do programa para Depurar a janela de Saída | Ativado | Exibe a saída do programa em uma janela separada do console e na janela de Saída do Visual Studio. Desmarque esta opção para mostrar a saída somente na janela do console separado. |
| Interromper a exceção SystemExit com código de saída zero | Desativado | Se definido, interrompe o depurador nessa exceção. Quando desmarcado, o depurador sai sem interromper. |
| Habilitar a depuração da biblioteca padrão do Python | Desativado | Torna possível intervir no código-fonte da biblioteca padrão durante a depuração, mas aumenta o tempo necessário para iniciar o depurador. |
| Mostrar o valor retornado da função | Ativado | Exibe os valores retornados de função na janela Locals , em seguida, passa uma chamada de função no depurador (F10) |
| Mostrar variáveis | Ativado | Exibe quatro grupos de variáveis a serem mostradas e como formatar a exibição (agrupar, ocultar, embutir). <ul style="list-style-type: none"> - Classe: o padrão é "Agrupar" - Protegido: o padrão é "Embutir" - Função: o padrão é "Agrupar" - Especial: o padrão é "Agrupar" |

Opções de análise

As seguintes opções estão disponíveis em **Ferramentas>Opções>Python>Análise:**

 Expandir a tabela

| Opção | Padrão | Descrição |
|--|--------------------------|--|
| Modo de diagnóstico | Somente arquivos abertos | Especifica quais arquivos de código o servidor de linguagem analisa em busca de problemas, incluindo Apenas arquivos do espaço de trabalho e Abertos . |
| Nível de log | Informações | Especifica o nível de registro em log a ser executado pelo servidor de linguagem. Os possíveis níveis de registro em log, no nível crescente de informações fornecidas, incluem Erro, Aviso, Informações e Rastreamento . |
| Verificação de tipo | Desativado | Especifica o nível da análise de verificação de tipo a ser executada: - Desativado : produza diagnóstico de importações/variáveis não resolvidas, mas não conduza análise de verificação de tipo - Básico : use regras sem tipo (todas as regras ativadas no nível Desativado) e regras básicas relacionadas à verificação de tipo - Estrito : use todas as regras de verificação de tipo na maior gravidade de erro, incluindo todas as regras ativadas nos níveis Desativado e Básico |
| Formato de importação | Absolute | Define o formato padrão ao importar módulos automaticamente, incluindo Absoluto ou Relativo . |
| Caminho de stubs | <Empty (vazio)> | Especifica um caminho para um diretório que contém stubs do tipo personalizados. Espera-se que os arquivos de stub de tipo para cada pacote estejam em seu próprio subdiretório. |
| Caminhos de pesquisa | <Empty (vazio)> | Especifica caminhos de pesquisa para resolução de importação. Aceita caminhos especificados como sequências e separados por vírgulas se houver vários caminhos, como <code>["path 1", "path 2"]</code> . |
| Caminhos Typeshed | <Empty (vazio)> | Especifica caminhos para que o Visual Studio use arquivos Typeshed personalizados em vez de sua versão agrupada. |
| Adicionar automaticamente caminhos de pesquisa comuns como 'src' | Ativado | Indica se os caminhos de pesquisa devem ser adicionados automaticamente com base em nomes predefinidos, como <code>src</code> . |
| Indexar bibliotecas instaladas de terceiros e arquivos de | Desativado | Especifica se o servidor de linguagem deve indexar arquivos de usuário e bibliotecas de |

| Opção | Padrão | Descrição |
|---|--------|--|
| usuário para recursos de linguagem, como importação automática, adição de importação, símbolos do espaço de trabalho etc. | | <p>terceiros instaladas na inicialização. O índice fornece um conjunto mais completo de símbolos em recursos, incluindo importações automáticas, correções rápidas, conclusões automáticas, dentre outros.</p> <ul style="list-style-type: none"> - Quando essa opção é configurada, o Visual Studio indexa os símbolos de nível superior dos pacotes instalados, como símbolos em tudo em package/_init__.py, com todos os símbolos de até 2.000 arquivos de usuário. - Quando essa opção não é configurada, o Visual Studio exibe símbolos referenciados ou usados em arquivos abertos anteriormente ou carregados pelo editor. |

Opções da Janela Interativa

As seguintes opções estão disponíveis em **Ferramentas>Opções>Python>Janelas Interativas:**

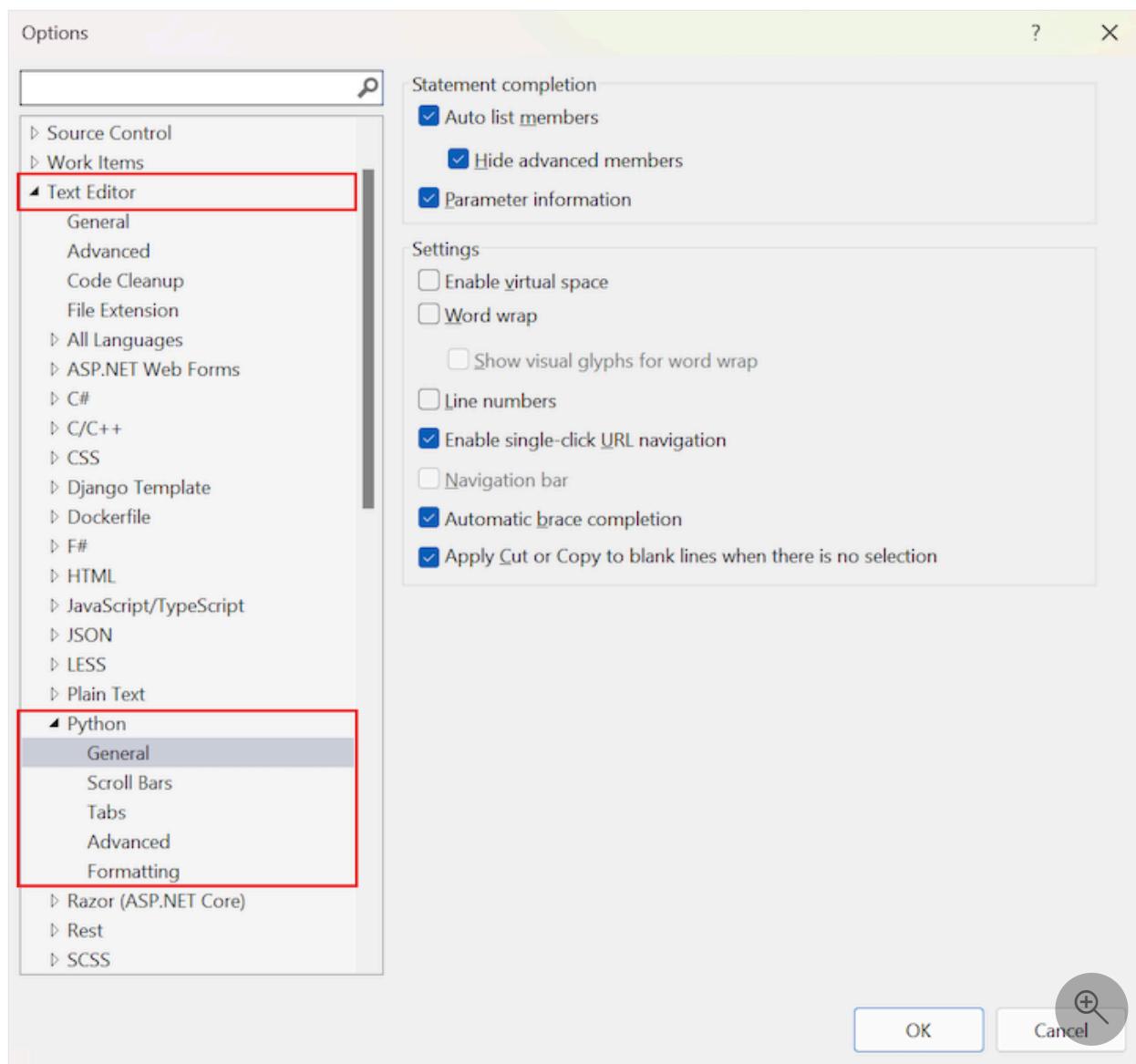
 Expandir a tabela

| Opção | Padrão | Descrição |
|---|---|---|
| Scripts | n/a | <p>Especifica uma pasta geral para scripts de inicialização a serem aplicados às Janelas Interativas de todos os ambientes. Para obter mais informações, confira Scripts de inicialização.</p> <p>Observação: esse recurso pode não funcionar na sua versão do Visual Studio.</p> |
| As setas para cima e para baixo navegam o histórico | Ativado | Usa as teclas de direção para navegar no histórico na janela Interativa . Desmarque essa configuração para usar as teclas de direção para navegar na saída da janela Interativa . |
| Modo de Conclusão | Avaliar somente expressões sem chamadas de função | O processo de determinar os membros disponíveis em uma expressão na Janela Interativa pode exigir a avaliação da expressão incompleta atual, que pode resultar em efeitos colaterais ou funções sendo chamadas várias vezes. A configuração padrão Avaliar somente expressões sem função chamadas exclui expressões que aparecem para chamar uma função, mas avaliada outras expressões. Por exemplo, ela avalia a instrução <code>a.b</code> , mas não a instrução <code>a().b</code> . Nunca avaliar expressões impede todos os efeitos colaterais, usando apenas o mecanismo IntelliSense normal para obter sugestões. Avaliar |

| Opção | Padrão | Descrição |
|---------------------------------------|------------|--|
| | | todas as expressões avalia a expressão completa para obter sugestões, independentemente de efeitos colaterais. |
| Ocultar sugestões de análise estática | Desativado | Quando definido, exibe apenas sugestões que são obtidas avaliando a expressão. Se combinado com o valor do Modo de Conclusão Nunca avaliar expressões, nenhuma conclusão útil será exibida na janela Interativa. |

Opções de editor de texto para Python

Em Editor de Texto > Python, há opções para barras de rolagem, guias e formatação, além de configurações gerais e avançadas:



Opções gerais de editor para Python

As seguintes opções estão disponíveis em **Ferramentas>Opções>Editor de Texto>Python>Geral**:

 Expandir a tabela

| Opção | Padrão | Descrição |
|--|------------|--|
| Listar membros automaticamente | Ativado | Defina essa opção para listar automaticamente os membros para conclusão das instruções de código. |
| Ocultar membros avançados | Ativado | Quando a opção Listar membros automaticamente estiver habilitada, defina essa opção para ocultar membros avançados das sugestões de conclusão. Membros avançados são aqueles usados com menos frequência do que outros. |
| Informações sobre parâmetros | Ativado | Quando essa opção é configurada, passar o mouse sobre os parâmetros mostra informações detalhadas, como a definição do item e links para a documentação. |
| Habilitar espaço virtual | Ativado | Quando essa opção é configurada, insere espaços no final de cada linha de código. Selecione essa opção para posicionar comentários em um ponto consistente ao lado do seu código. O modo Espaço Virtual está habilitado no modo Seleção de Coluna . Quando o modo Espaço Virtual não está habilitado, o ponto de inserção é movido do final de uma linha diretamente para o primeiro caractere da próxima. Observação: essa opção é influenciada pela configuração global Editor de Texto>Todas as Linguagens>Geral>Habilitar espaço virtual . Se a configuração global não estiver habilitada, essa opção não pode ser habilitada no nível da linguagem. |
| Quebra automática de linha | Desativado | Defina essa opção para permitir que longas linhas de código sejam quebradas com base na largura do visor dos editores. |
| Mostrar glifos visuais para quebra automática de linha | Desativado | Quando a opção Quebra automática de linha estiver habilitada, defina essa opção para mostrar glifos visuais. |
| Números de linha | Desativado | Defina essa opção para mostrar números de linha na margem esquerda do editor para cada linha de código. |
| Habilitar navegação de URL com um só clique | Ativado | Quando essa opção é configurada, você pode dar um clique simples em uma URL para navegar ao local de destino. |

| Opção | Padrão | Descrição |
|---|------------|--|
| Barra de navegação | Desativado | <p>Defina essa opção para habilitar as caixas suspensas na parte superior da janela de código. Esses campos ajudam a navegar até o código em uma base de código onde você pode escolher um tipo ou membro ao qual ir diretamente.</p> <p>Observação: essa opção é influenciada pela configuração global Editor de Texto>Todas as Linguagens>Geral>Habilitar barra de navegação. Para obter mais informações, confira Navegar pelo código>Barra de navegação.</p> |
| Preenchimento automático de chaves | Ativado | Quando essa opção é configurada, o Visual Studio adiciona automaticamente a chave de fechamento a qualquer chave aberta à medida que o código é inserido. |
| Aplicar Cortar ou Copiar a linhas em branco quando não houver nenhuma seleção | Ativado | Por padrão, o Visual Studio corte ou copia toda a linha de código quando não há seleção explícita. Use essa opção para habilitar ou desabilitar esse comportamento de Cortar ou Copiar quando invocado em linhas em branco. |

Para obter mais informações, confira [Caixa de diálogo Opções: editor de texto > geral](#).

Opções avançadas de editor de Python

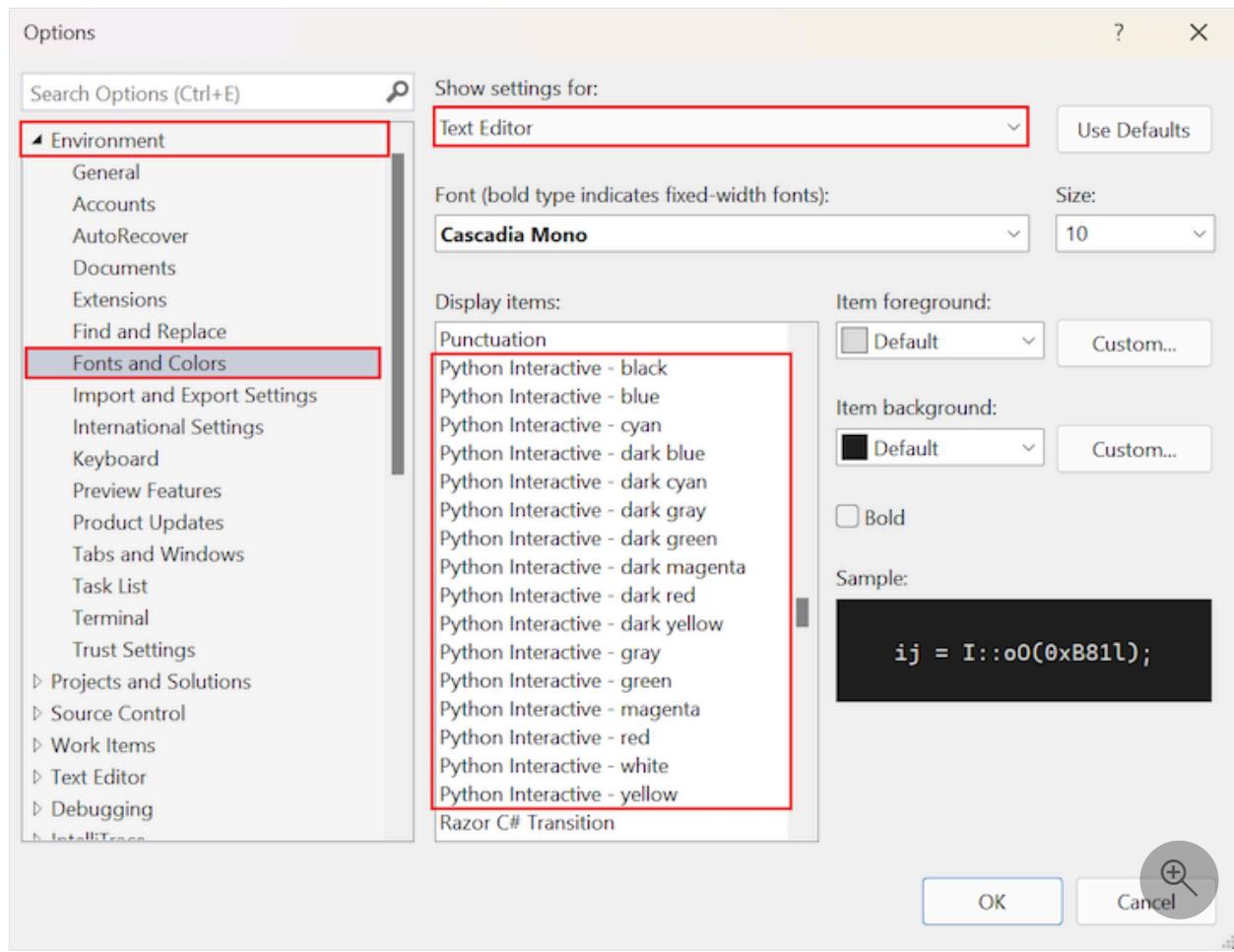
As seguintes opções estão disponíveis em **Ferramentas>Opções>Editor de Texto>Python>Avançado**:

 Expandir a tabela

| Opção | Padrão | Descrição |
|--|------------|--|
| Oferecer conclusões de importação automática | Ativado | Quando essa opção é configurada, o Visual Studio oferece importações automáticas na conclusão. |
| Adicionar grupos a funções automaticamente | Desativado | Quando essa opção é configurada, o Visual Studio adiciona automaticamente grupos a funções à medida que o código é inserido no editor. |

Opções de Fontes e Cores

Outras opções do Python estão disponíveis em **Ambiente>Fontes e Cores** quando o grupo **Editor de Texto** está configurado para **Python**:



Os nomes das opções de Python são prefixados com "Python" e são autoexplicativas. A fonte padrão para todos os temas de cores do Visual Studio é 10 pt Consolas regular (não está em negrito). As cores padrão variam de acordo com o tema. Normalmente, você altera uma fonte ou cor para facilitar a leitura do texto.

Comentários

Esta página foi útil?

Yes

No