

# Programação

Victor Machado da Silva, MSc  
victor.silva@professores.ibmec.edu.br



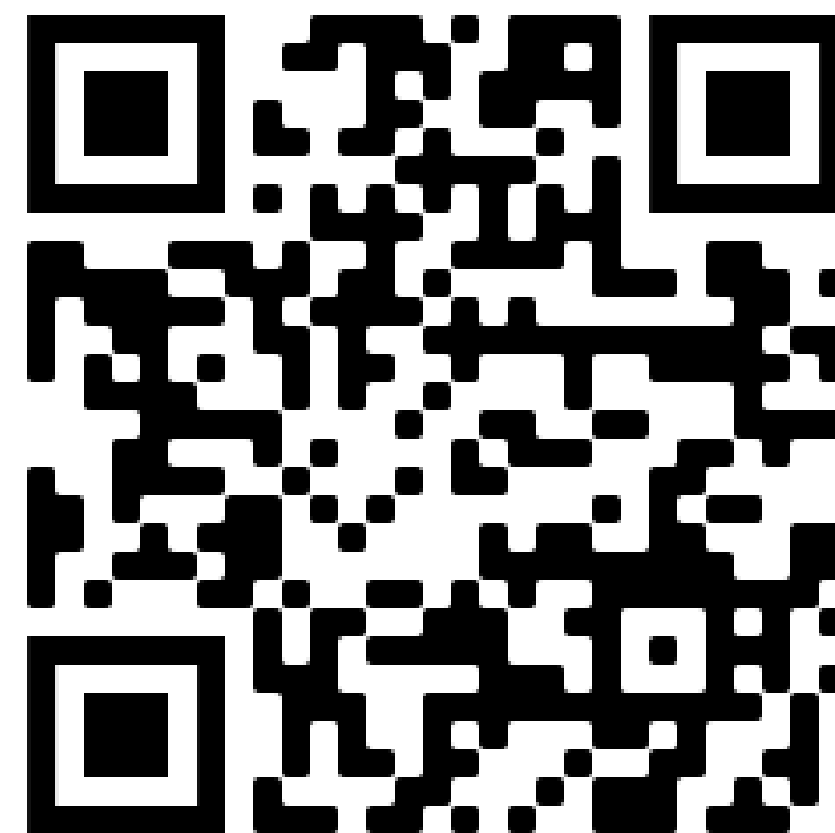
# Índice

- [Apresentação do curso](#)
- [Por que aprender a programar?](#)
- [Configurando o ambiente Python](#)
- [Instalando e configurando IDEs](#)
- [Utilizando o PyCharm](#)
- [Utilizando o VSCode](#)
- [Configurando o pylint](#)
- [Instalando pacotes pelo PyPI](#)
- [Noções básicas de OO](#)

# Apresentação do curso

# Apresentação do curso

- Contato: victor.silva@professores.ibmec.edu.br
- Aulas às segundas e quartas-feiras, de 9:50 às 11:40
- Grupo no Whatsapp: <https://chat.whatsapp.com/IFDRIyeZewh72bcJ0FQVj2>
- Site com materiais: <http://victor0machado.github.io>



# Apresentação do curso

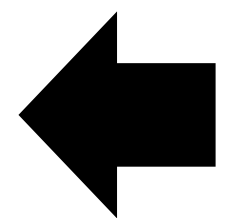
- Avaliação
  - Proporção:
    - AC (20%): Listas de exercícios e atividades em sala, individuais
    - AP1 e AP2 (40% cada): Projetos de disciplina, em grupo
  - AS será um trabalho individual, que substituirá a menor nota entre AP1 e AP2.
  - Para AP1 e AP2, grupos de no mínimo 4 e no máximo 5 pessoas.

# Sugestões de materiais para estudo

- Python é uma linguagem intuitiva para o aprendizado, porém é importante termos à mão livros, apostilas e outros materiais para auxiliar os estudos. Abaixo encontram-se algumas sugestões:
  - Documentação oficial em Python: <https://docs.python.org/pt-br/3/index.html>
  - Stack Overflow: <https://stackoverflow.com/questions/tagged/python>
  - Artigos no Medium.com: <https://medium.com/search?q=python>

# Sugestões de materiais para estudo

- Canais interessantes no Youtube sobre Python e programação:
  - Programação Dinâmica: <https://www.youtube.com/c/ProgramacaoDinamica/>
  - Curso em Vídeo: <https://www.youtube.com/c/CursoemVideo/>
  - Sentdex (em inglês): <https://www.youtube.com/c/sentdex>
  - Filipe Deschamps: <https://www.youtube.com/c/FilipeDeschamps>
  - DevMedia: <https://www.youtube.com/c/DevmediaBrasil>



# Por que aprender a programar?



# O que significa “programar”?

- Programar um computador significa fornecer à máquina um conjunto de instruções que indica o que você quer que ele faça.
  - Quando você pressiona o botão “enviar” no seu aplicativo de e-mail, está comandando a máquina para que ela produza um determinado resultado. Ela reage a esse comando executando um conjunto de instruções previamente programadas pelos criadores do aplicativo, as quais realizam a tarefa de enviar um e-mail.
- A programação passa então por acessar essas características internas da máquina e ser capaz de usá-las
- Programar significa escrever instruções que sejam executadas diretamente pela máquina, sem utilizar aplicações intermediárias preconcebidas como planilhas ou browsers de internet.

# O que significa “programar”?

- A programação se resume a uns poucos conceitos fundamentais:
  - *Entrada de dados:* define de que forma a máquina obtém informações, que podem ser recebidas via teclado, arquivos, rede, mouse ou outros dispositivos. Essas informações tanto podem estar sendo produzidas por usuários humanos quanto por outros dispositivos.
  - *Saída de dados:* define como a máquina apresenta respostas ao mundo externo, que podem ser exibidas através de um monitor, impressora ou outros dispositivos. Da mesma forma que as entradas, as saídas podem ser enviadas usuários humanos ou a outros dispositivos.

# O que significa “programar”?

- A programação se resume a uns poucos conceitos fundamentais:
  - *Processamento*: define como os dados de entrada são transformados e combinados com outras informações eventualmente já armazenadas pela máquina, ou obtidas por ela, para produzir as saídas. O processamento segue uma lógica de execução que é conhecida como “algoritmo”. Um algoritmo é basicamente uma lista finita de instruções que resolve um problema específico e se estrutura a partir de:
    - *Comandos*: instruções individuais que o programador dá à máquina e que ela é capaz de executar;
    - *Seleção*: estruturas que permitem à máquina decidir quando executar uma determinada lista de comandos ou outra;
    - *Repetição*: estruturas que permitem à máquina repetir uma lista de comandos um número fixo de vezes ou até que uma determinada condição seja obtida.

# Linguagens de programação?

- A forma como o programador transmite as instruções definidas no algoritmo para a máquina é dada através de um código, escrito utilizando uma linguagem de programação
- Existem basicamente dois tipos de linguagem de programação:
  - As de *baixo nível* são aquelas cujos comandos podem ser executados diretamente pelo hardware do computador, ou seja, pelo microprocessador da máquina. Esse tipo de linguagem usualmente é utilizado por programadores altamente especializados em situações muito particulares ligadas ao controle das funções mais básicas da máquina;
  - A maioria dos programas hoje em dia são escritos em linguagens de *alto nível*, ou seja, linguagens que possuem comandos muito mais fáceis de serem entendidos por um ser humano. Essas linguagens usualmente produzem programas muito mais fáceis de ler.
- [Evolução das linguagens de programação](#)



# Linguagens de programação?

- O problema das linguagens de alto nível é que o hardware não consegue executar diretamente esses comandos. Assim, um programa especial deve ser usado para traduzir os comandos de alto nível em comandos de baixo nível executáveis pela máquina.
- Existem basicamente duas formas principais de fazer essa tradução: através de programas compiladores e programas interpretadores:
  - Um *compilador* é um programa que traduzirá a totalidade dos comandos escritos em linguagem de alto nível gerando um programa completo traduzido para a linguagem de baixo nível. Esse novo programa traduzido pode então ser diretamente executado pela máquina. Exemplos: C, C++, Pascal, Fortran.
  - Os *interpretadores* normalmente tomam um comando de vez na linguagem de alto nível e interpretam o seu significado, executando diretamente os comandos de alto nível em uma máquina virtual, que simula um computador de alto nível. Exemplos: Python, PHP, Ruby, Lua.

# Linguagens de programação?

- Existe uma diferença fundamental entre a linguagem humana (ou natural) e a linguagem de computadores. Os seres humanos são capazes de compreender expressões mesmo que elas não estejam escritas corretamente.

Olá! Td bem c vc?

- Um tópico fundamental em programação é:

*Programas são escritos em uma linguagem formal. Cada letra e cada sinal matemático ou de pontuação têm um significado muito preciso. Mudar qualquer um destes sinais pode, muitas vezes, mudar o sentido do programa, ou até fazê-lo parar de funcionar.*

```
1  for cont in range(10):  
2  |  print(str(cont))  
3  print(str(cont))
```

```
1  for cont in range(10):  
2  |  print(str(cont))  
3  |  print(str(cont))
```

# O que é um programa?

- Um *programa* consiste em um texto escrito em uma determinada linguagem de programação, seja de alto ou baixo nível, que por sua vez é processada de forma a comunicar ao computador comandos que devem ser executados.
- Um programa pode conter um ou mais algoritmos.
- O texto escrito do programa (ou código) pode ser armazenado no computador com extensões que variam conforme a linguagem de programação que estiver sendo utilizada.
- Em particular, a linguagem Python aceita diversas extensões, sendo a mais comum (e utilizada nesta disciplina) a extensão **.py**.



# Por que Python?

- Python foi concebido no final da década de 1980, por Guido van Rossum, na Holanda
- A linguagem foi batizada em homenagem ao programa de TV britânico *Monty Python's Flying Circus*, que fazia muito sucesso na época em boa parte do mundo
- A linguagem foi desenvolvida com o objetivo de ser simples de se desenvolver, e com uma estrutura semântica e sintática intuitiva e natural



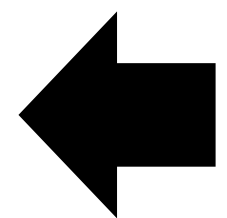


# Por que Python?

- Características do Python:
  - Linguagem de propósito geral
  - Fácil e intuitiva
  - Multiplataforma
  - É possível começar a programar de forma simples, sem instalar inúmeros pacotes
  - Livre
  - Organizada
  - Orientada a objetos
  - Inúmeras bibliotecas à disposição
  - Extensa comunidade, com vasta documentação online

# Por que Python?

- Principais áreas de atuação:
  - Inteligência Artificial
  - Biotecnologia
  - Computação 3D
  - Ciência de Dados
  - Internet das Coisas



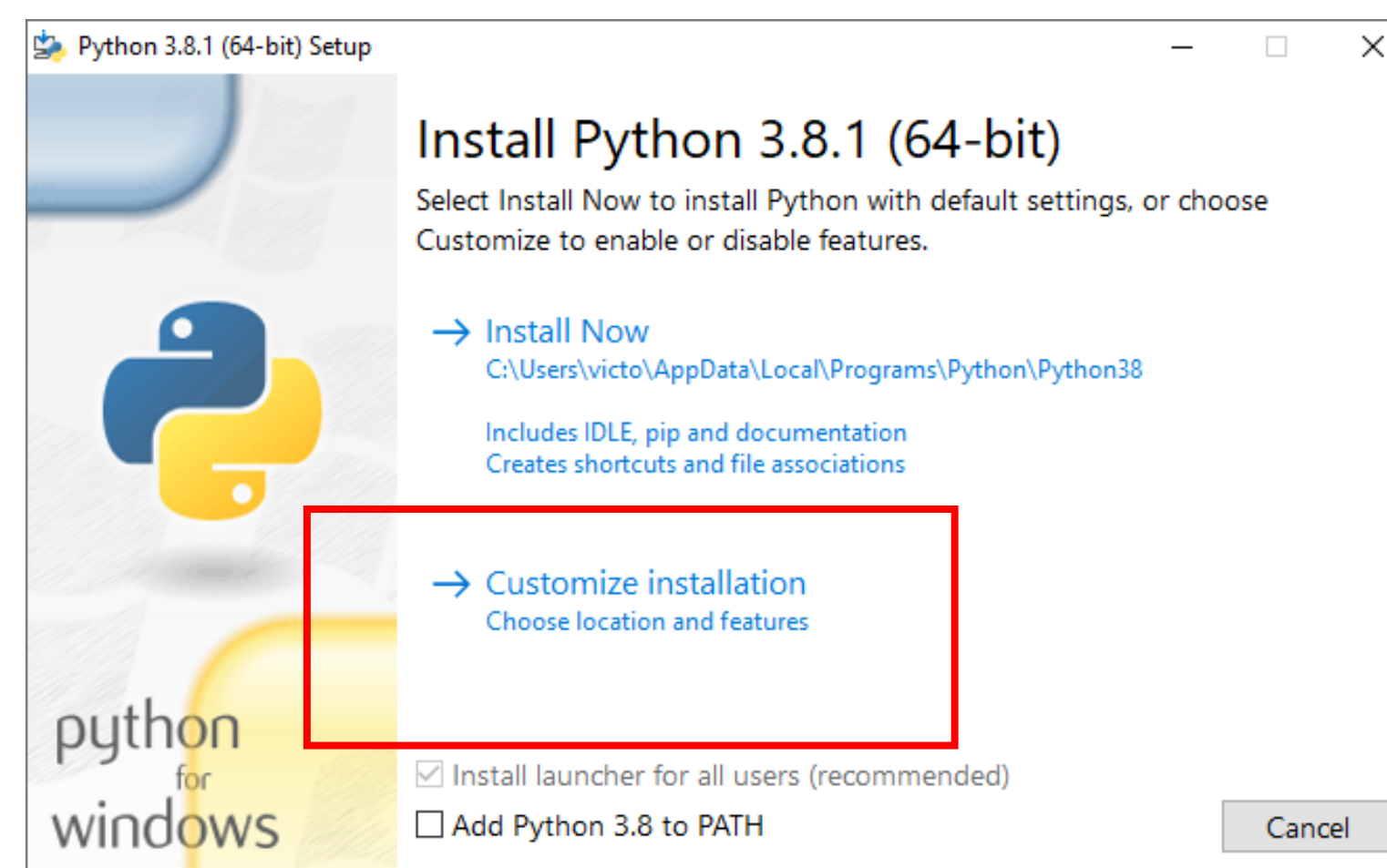
# Configurando o ambiente Python

# O que é Python?

- Python é uma linguagem de programação de alto nível, lançada por Guido van Rossum em 1991. Atualmente é uma das linguagens de uso mais abrangentes no mundo todo, principalmente nas áreas de Data Science e em aplicações de *back-end*, ou seja, de processamento de dados que não interagem diretamente com o usuário final.
- Diversas organizações utilizam Python atualmente:
  - Google
  - Yahoo!
  - NASA
  - AirCanada

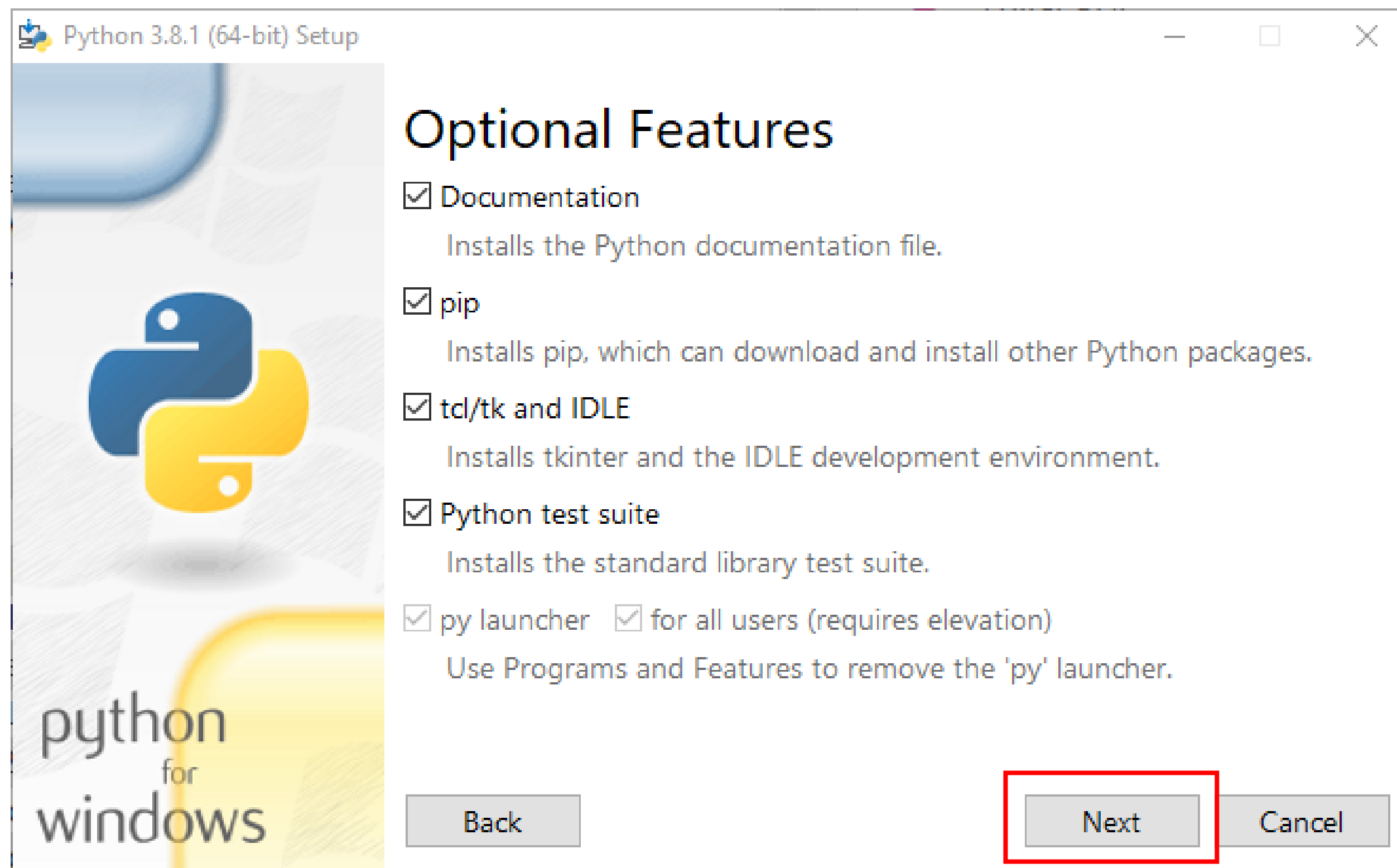
# Como instalar Python?

- Neste curso podemos trabalhar com qualquer versão recente do Python, 3.9 ou superior. Para baixar o instalador para Windows, clique [neste link](#). O download do instalador para macOS se encontra [neste link](#).
- Este curso focará no uso do Python para Windows. Para o uso no macOS, veja no [site oficial da linguagem](#) informações particulares.
- Ao clicar no instalador, selecione a opção “Customize installation”.



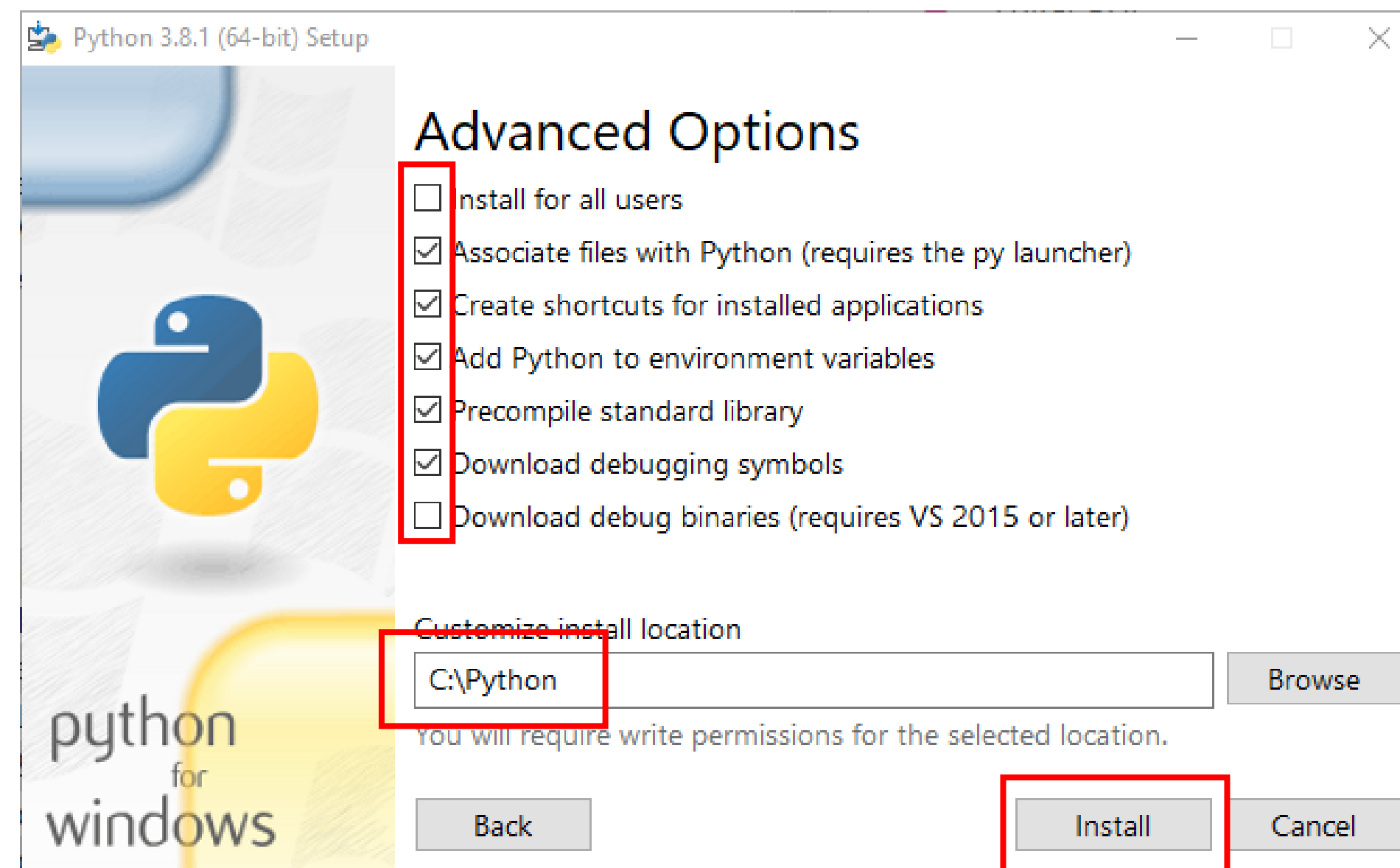
# Como instalar Python?

- Na tela “Optional Features”, clique em “Next”.




# Como instalar Python?

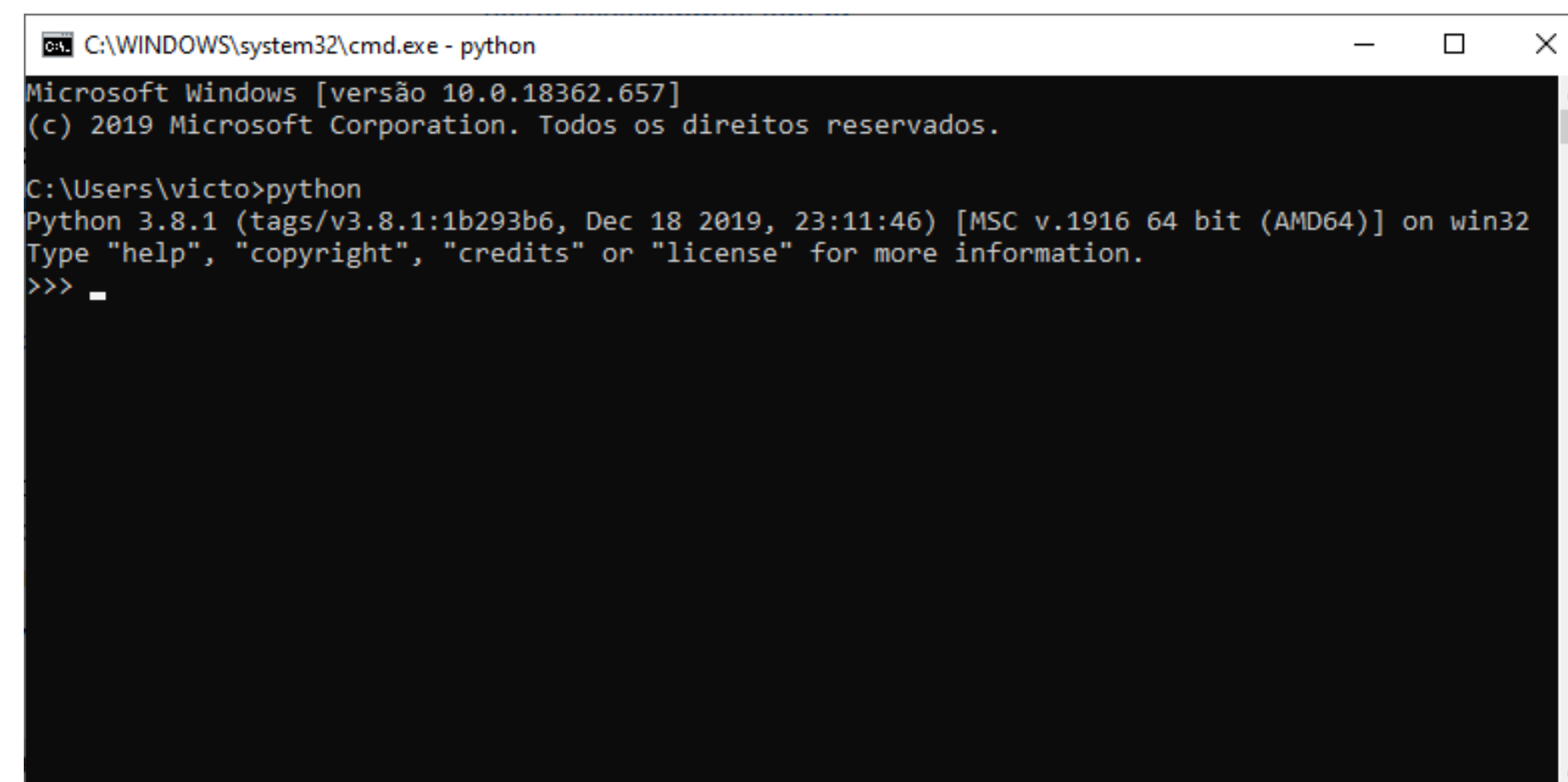
- Na tela “Advanced Options”, deixe as caixas de opções marcadas conforme a imagem abaixo.
- No campo “Customize install location”, escolha um caminho de fácil acesso. O caminho sugerido é “C:\Python”.
- Com tudo pronto, clique em “Install”, e conclua após a mensagem de sucesso.





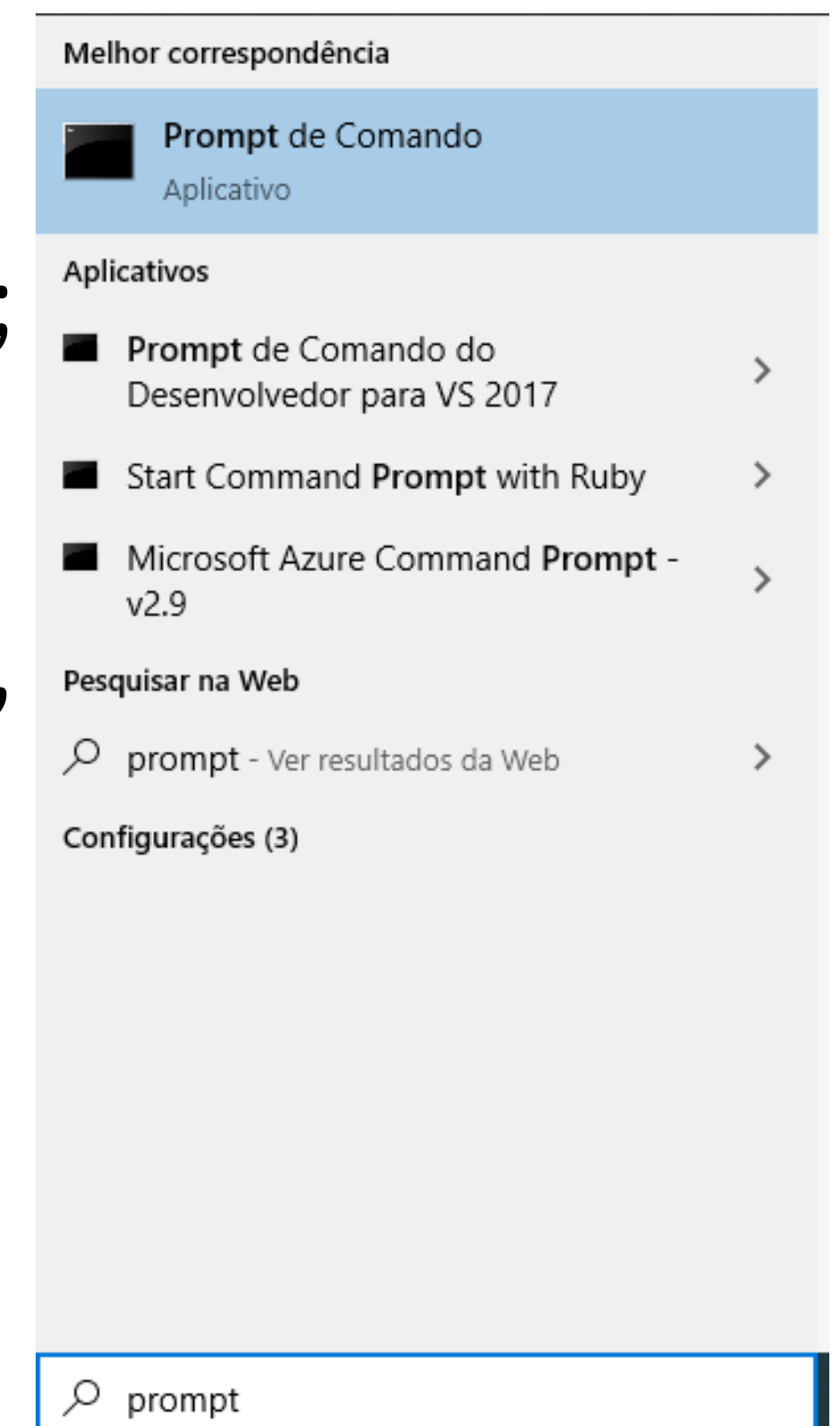
# Como instalar Python?

- Para conferir se a instalação foi bem sucedida, faça os seguintes passos:
  - Clique no botão do Windows ;
  - Digite a caixa de pesquisa **prompt de comando**, e abra o programa;
  - Na janela que abrir, digite o comando **python** e pressione Enter;
  - O Windows deve inicializar um editor de Python na mesma janela, como mostrado abaixo.



```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [versão 10.0.18362.657]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

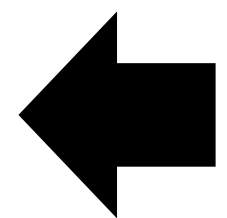
C:\Users\victo>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```





# Algumas dicas iniciais após instalar o Python

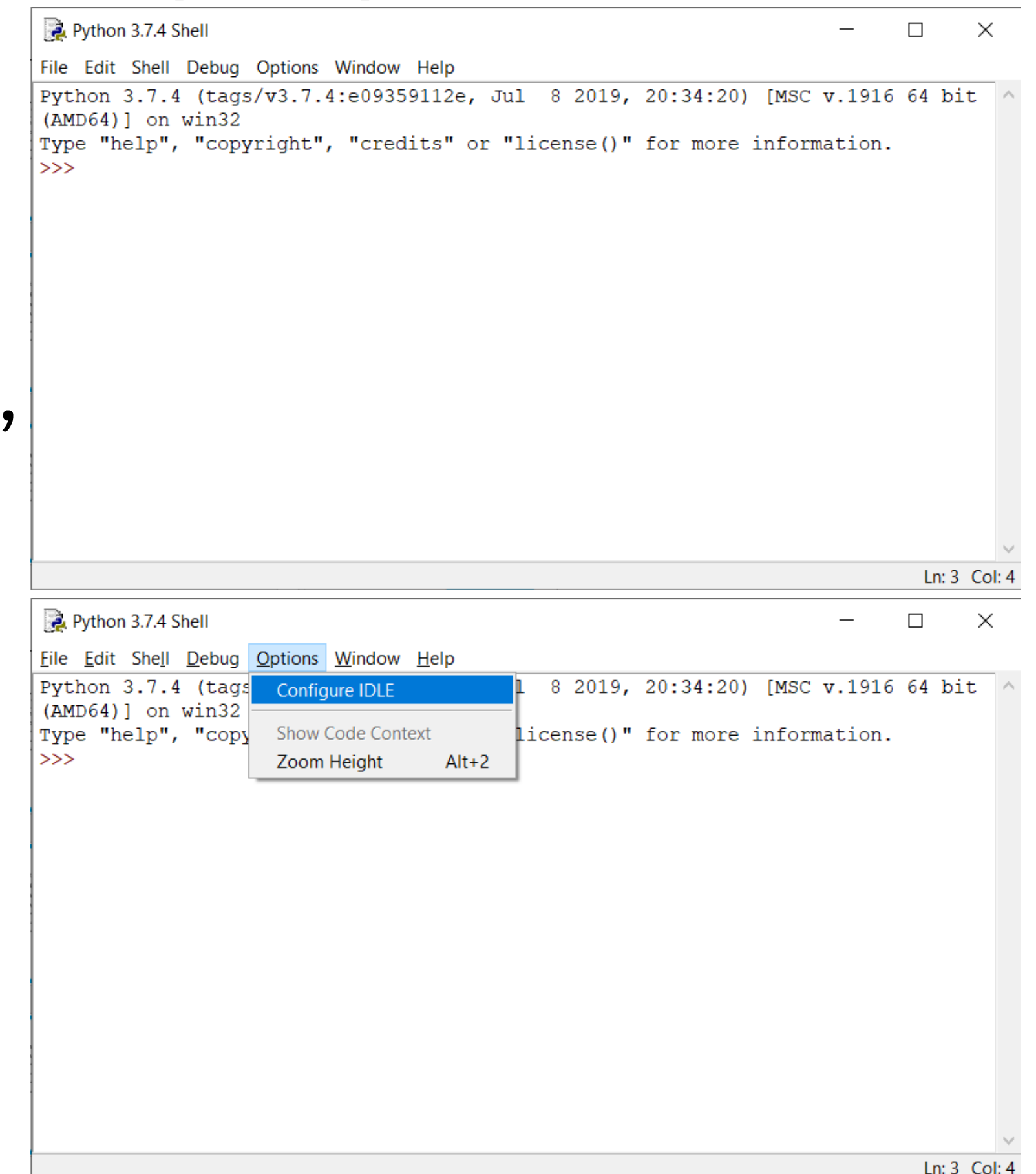
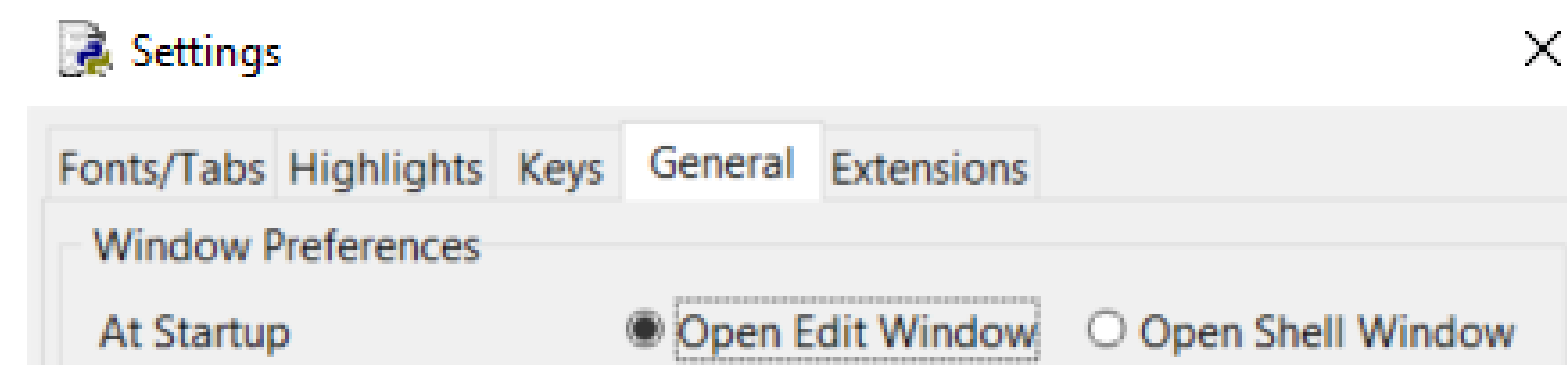
1. Antes de abrir o programa, abra o Windows Explorer, clique em **Este Computador** e, em seguida, no drive do seu computador (C: ou D:, dependendo da sua máquina);
2. Neste diretório, crie uma pasta chamada **Projetos**. Esta pasta será usada para armazenar todos os seus projetos de software;
3. Dentro da pasta de projetos, crie a pasta da disciplina (p.ex., **algoritmos**);
4. Evite utilizar caminhos muito longos (p.ex., C:\Users\12304010\Projetos\Nome-da-pessoa\Documentos\etc...) ou incluir espaços no caminhos (p.ex., C:\Victor Machado). O primeiro é muito trabalhoso para utiliza-lo recorrentemente, e o segundo pode causar alguns problemas na execução do código;
5. Sempre que criar arquivos Python, comece o nome do arquivo com uma letra (p.ex., **main.py**, **app.py**, **aula.py**). Evite usar números, espaços ou acentos nos nomes dos arquivos.



# Instalando e configurando IDEs

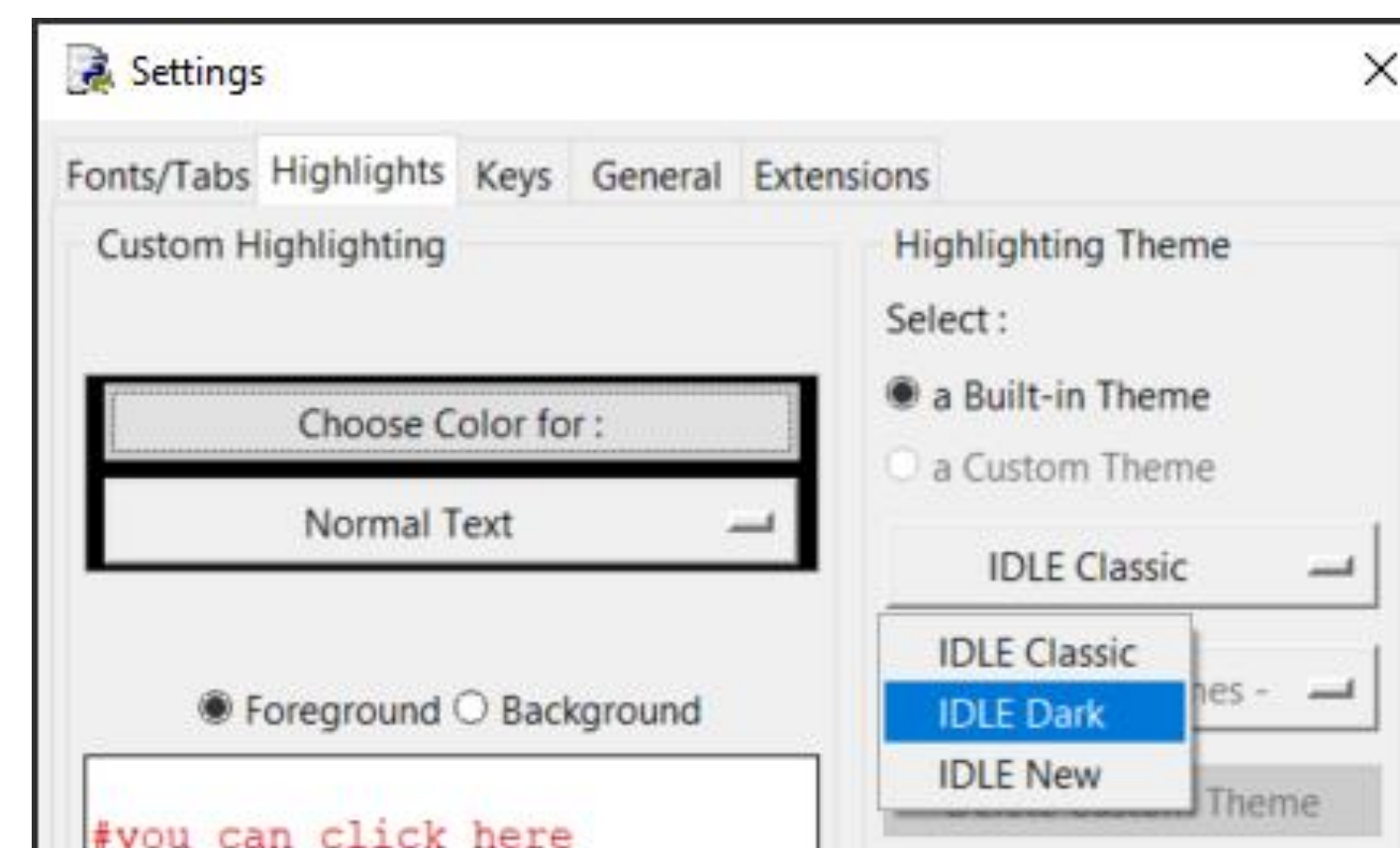
# Configurando o IDLE

- Abra IDLE utilizando o ícone do Windows e procurando pelo programa
- O programa abre no modo **Shell**, que é a janela de execução do código. Usamos essa tela apenas para checar os resultados ou quando queremos executar códigos de uma linha (testar uma função, por exemplo)
- Para abrir o editor automaticamente, vá no menu **Options > Configure IDLE**
- Na aba **General**, em **Windows Preferences**, marque a opção **Open Edit Window**. Clique em **Ok** e reinicie o IDLE



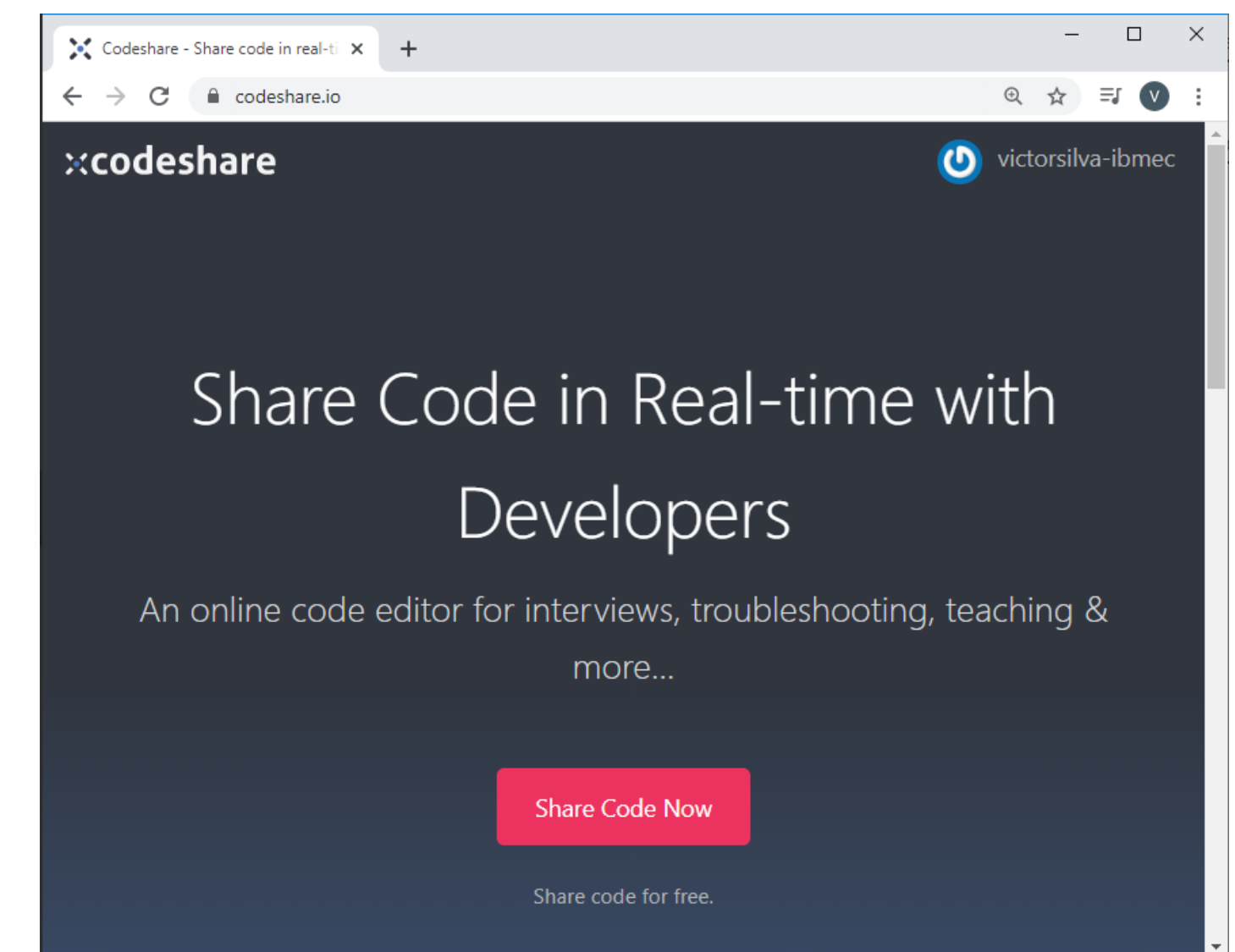
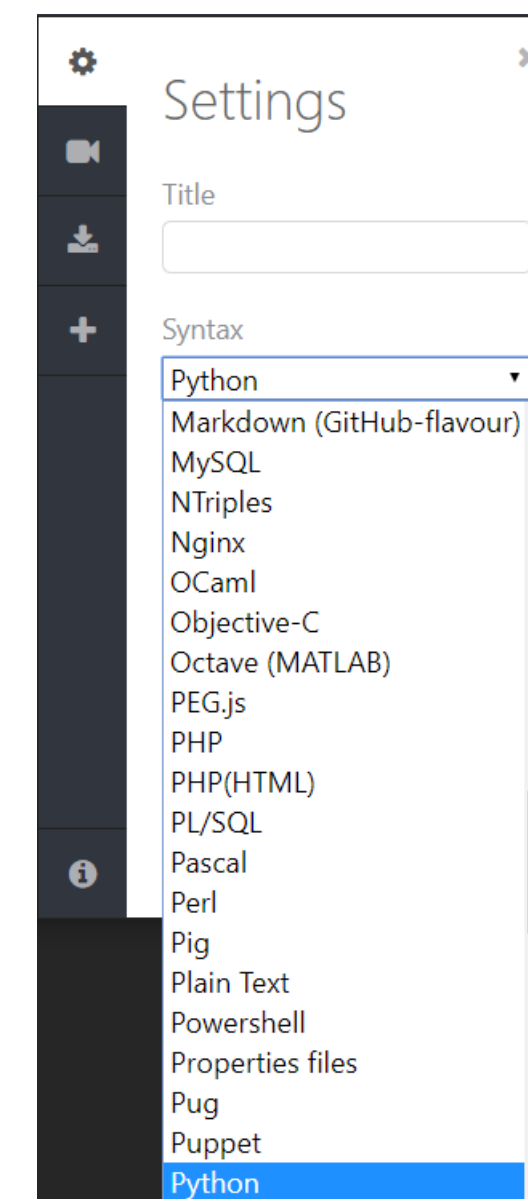
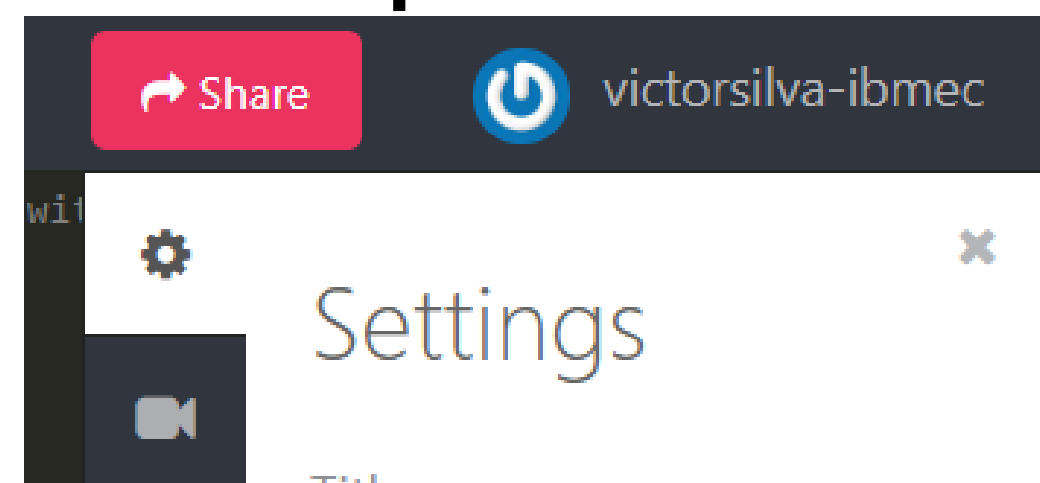
# Configurando o IDLE

- Normalmente é usual programadores trabalharem com editores de texto que tenham um fundo escuro, o que prejudica menos a visão
- Na mesma tela de **Configurações**, vá para a aba **Highlights**, e na coluna da direita, em **Highlight Theme**, clique em **IDLE Classic** e selecione a opção **IDLE Dark**
- Clique em OK para mudar o tema para escuro



# Usando o CodeShare

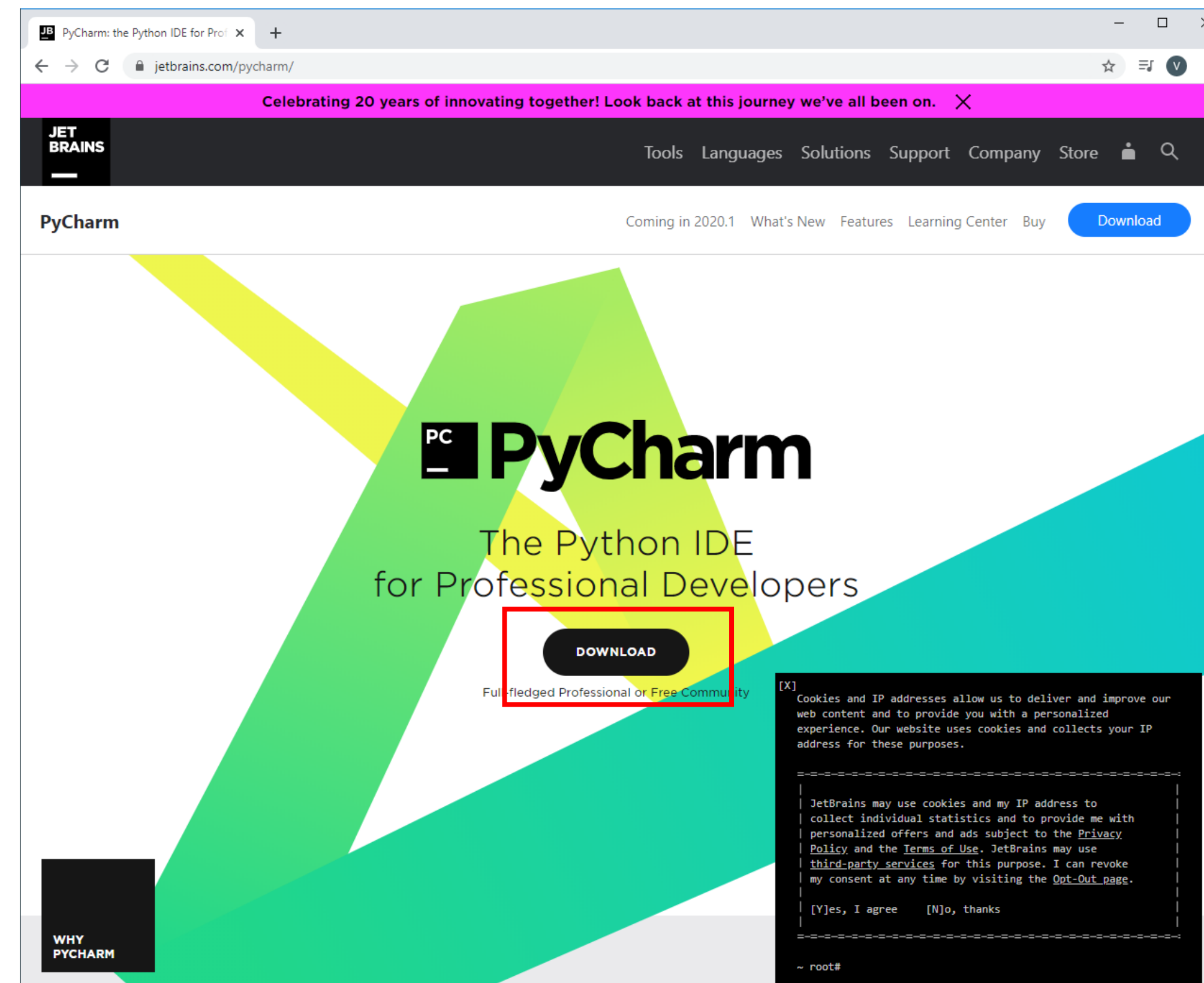
- Acesse <http://codeshare.io> e faça o cadastro
- Tendo cadastrado, na tela inicial clique em “Share Code Now”
- Um editor de texto vai aparecer. Na coluna da direita, clique na engrenagem, e em **Syntax** marque a opção **Python**
- Para compartilhar, clique em **Share**, no canto superior da janela
- O CodeShare vai liberar um link para ser usado por outras pessoas





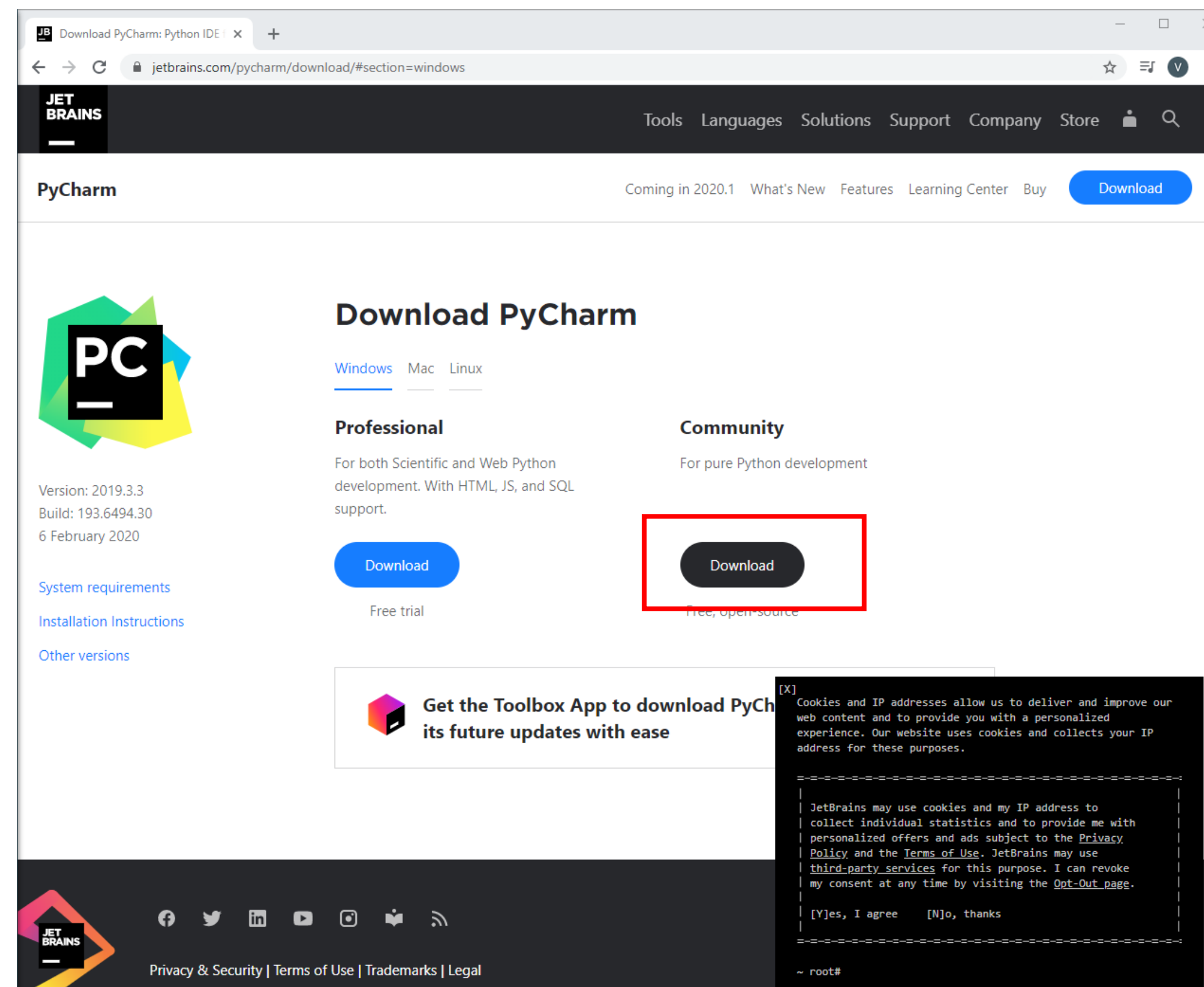
# Instalando e configurando o PyCharm

- O PyCharm é um dos editores (ou IDEs) mais usados para a programação de aplicações em Python. Para baixar e instalar, primeiro acesse a página <https://www.jetbrains.com/pycharm/> e clique em “Download”.



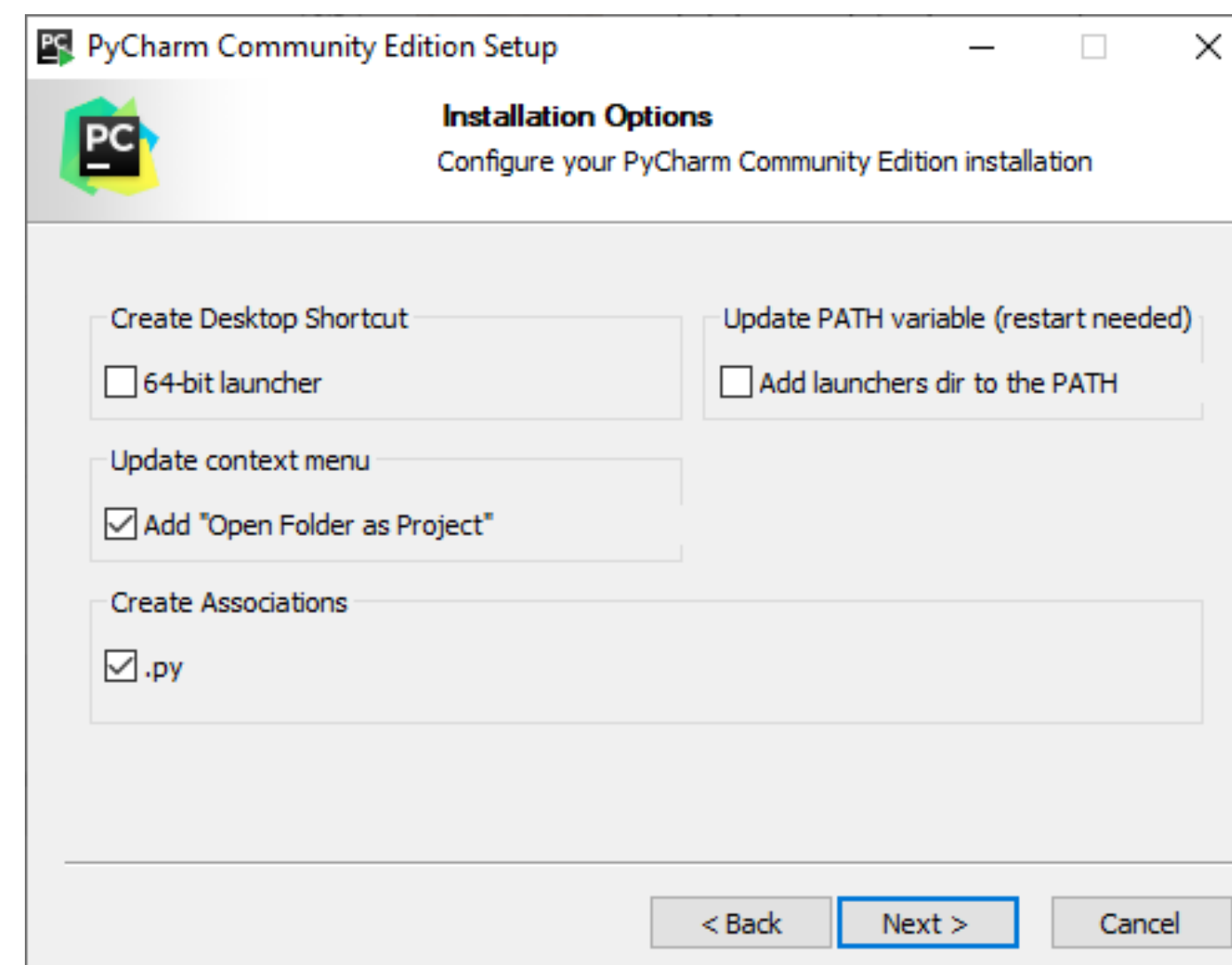
# Instalando e configurando o PyCharm

- Escolha o seu sistema operacional (Windows, Mac ou Linux – vamos trabalhar com Windows) e baixe a versão “Community”. O download deve começar automaticamente.



# Instalando e configurando o PyCharm

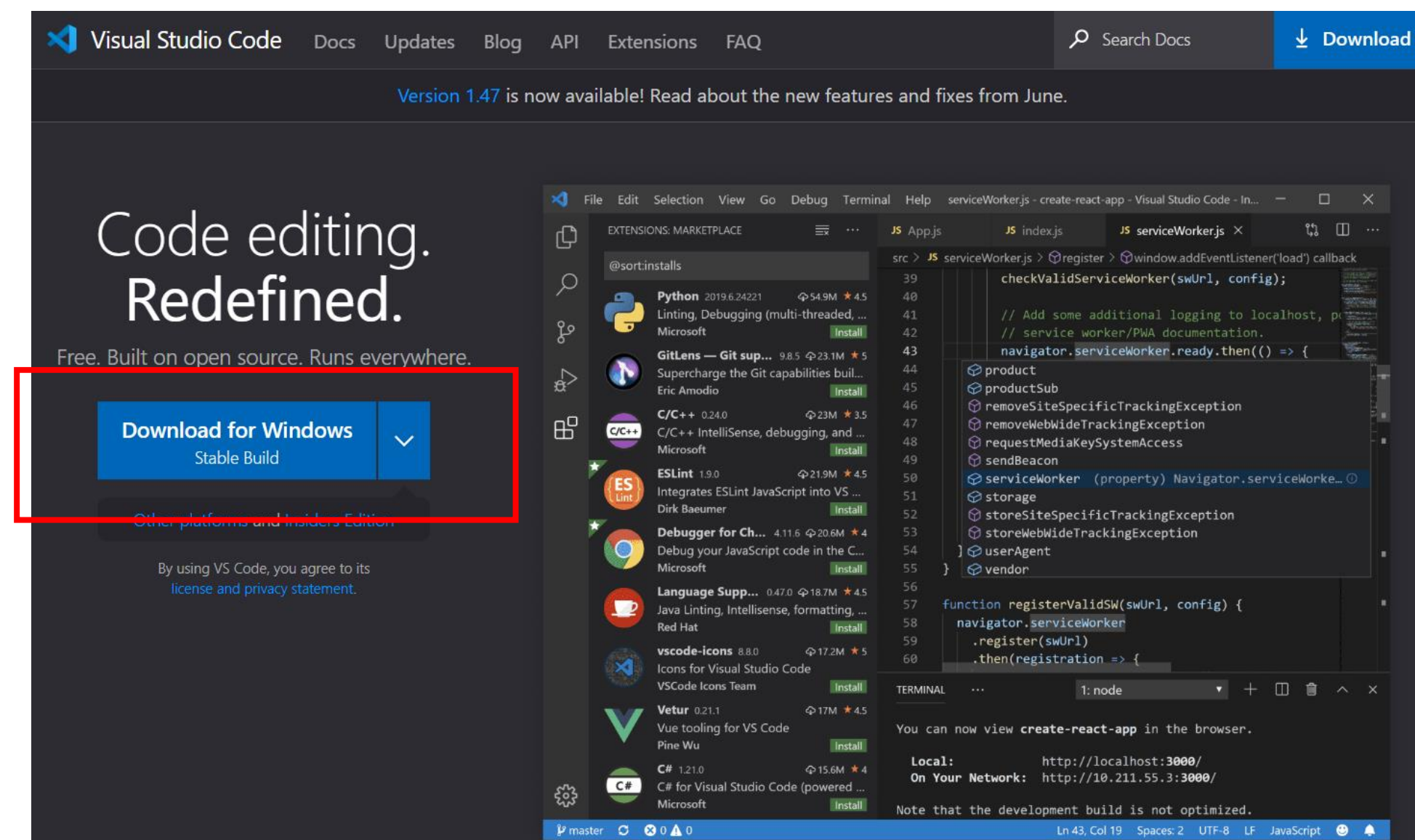
- Abra o instalador do PyCharm e clique em “Next”;
- Na tela de escolha do caminho de instalação, clique em “Next”;
- Na tela seguinte, marque as opções indicadas na imagem abaixo e clique em “Next”;
- Na tela seguinte, clique em “Install” para começar a instalação.





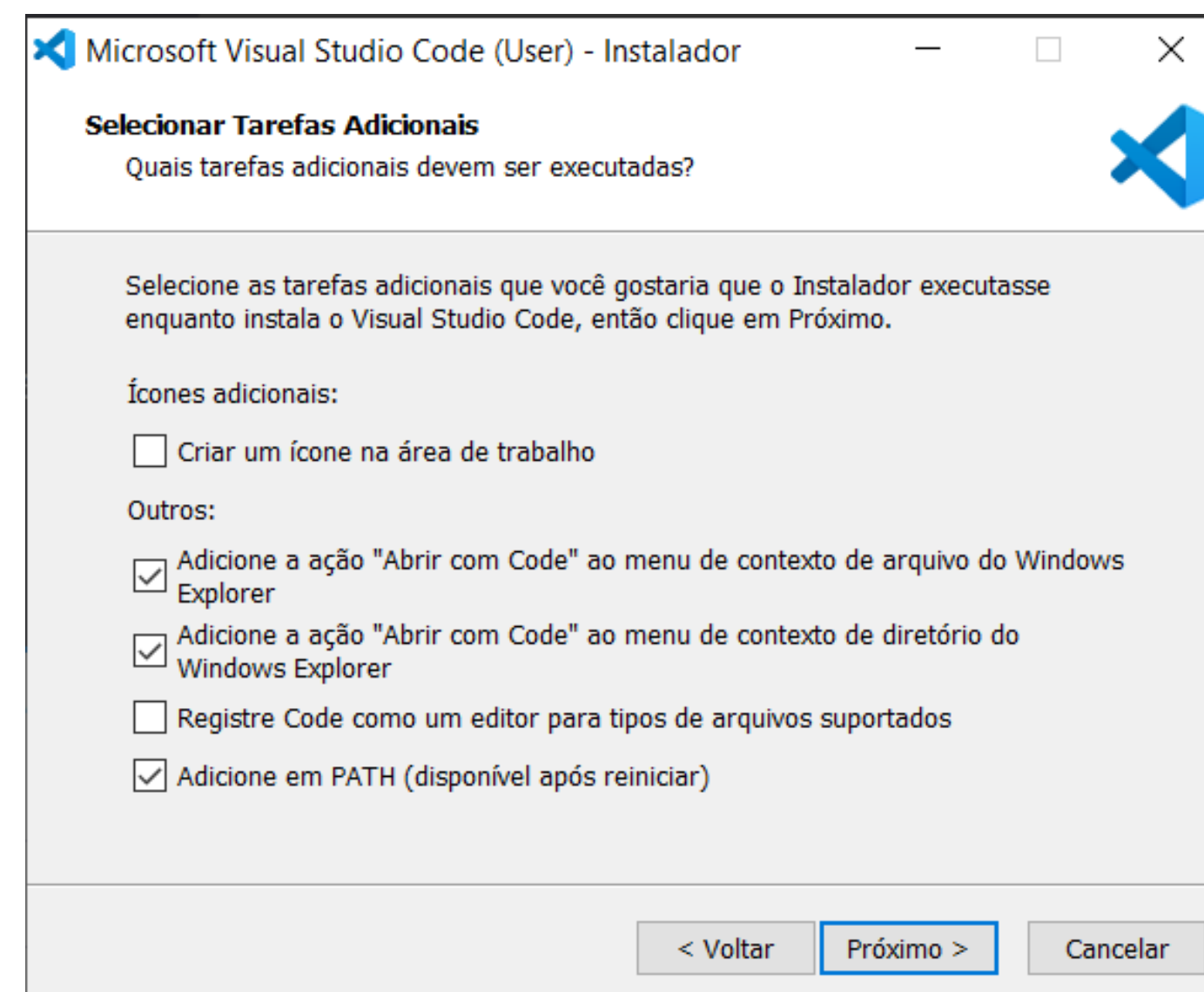
# Instalando e configurando o VSCode

- O **VSCode** é um dos IDEs recentes mais famosos para basicamente qualquer linguagem de programação. Possui inúmeras extensões que facilitam e customizam o editor para a necessidade de cada programador. Para baixar e instalar, primeiro acesse a página <https://code.visualstudio.com/> e clique em “Download for Windows”. Clicando na seta à direita existem opções para MAC e Linux.



# Instalando e configurando o VSCode

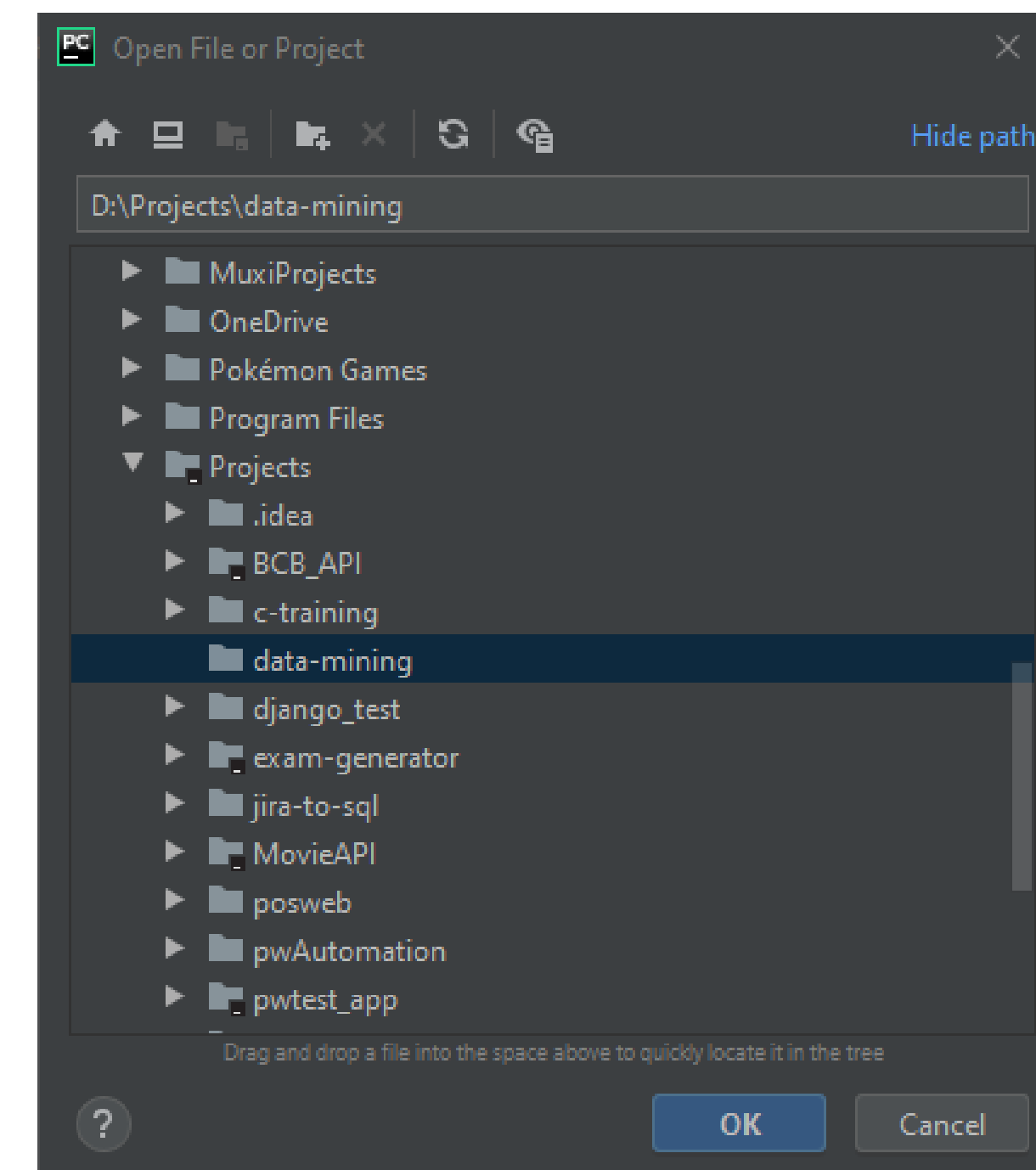
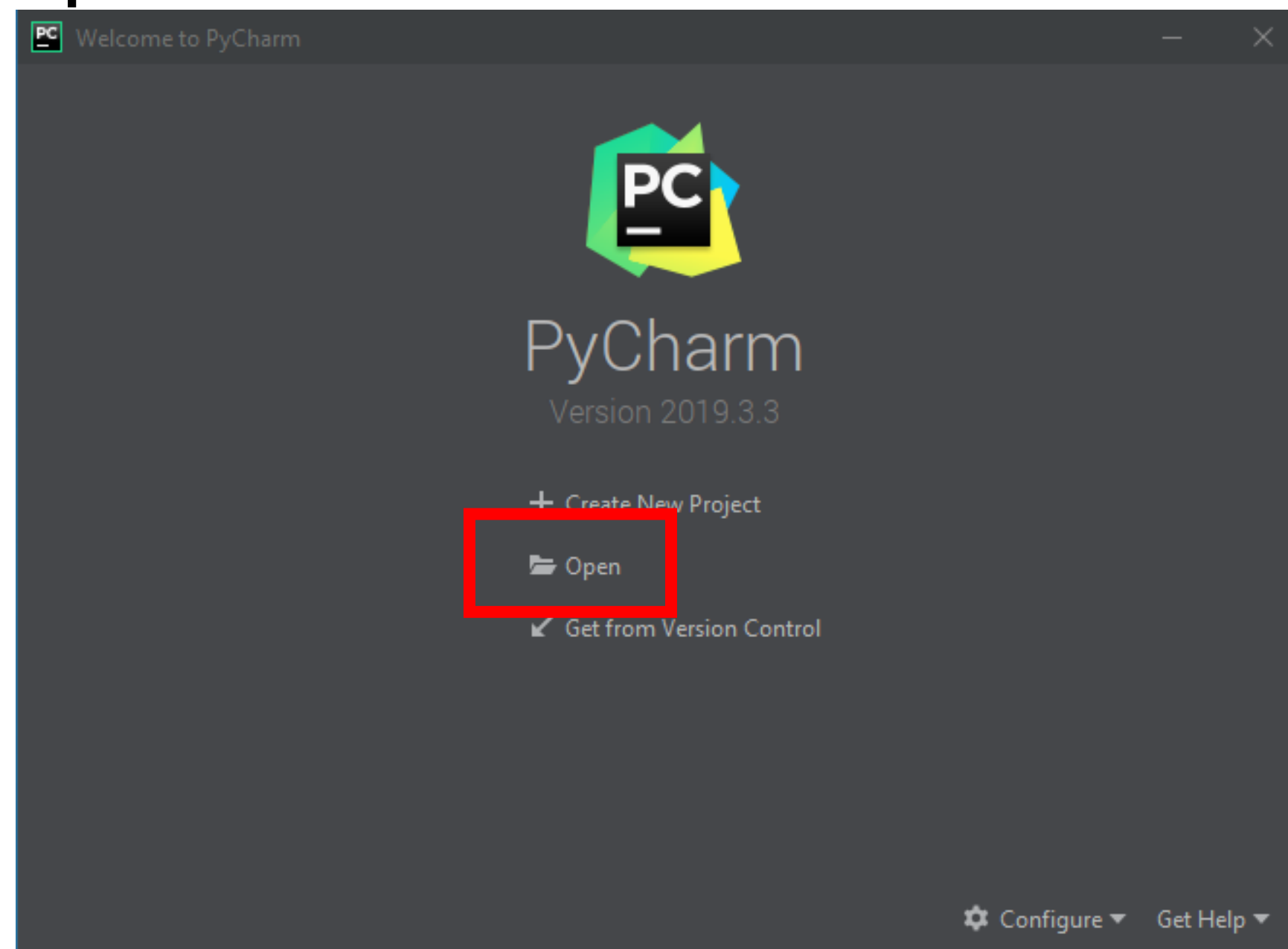
- Abra o instalador, e após aceitar o acordo de licença e clicar em “Próximo”, clique em “Próximo” novamente até chegar na tela “Selecionar Tarefas Adicionais”
- Marque as opções abaixo e clique em “Próximo” e depois em “Instalar”



# Utilizando o PyCharm

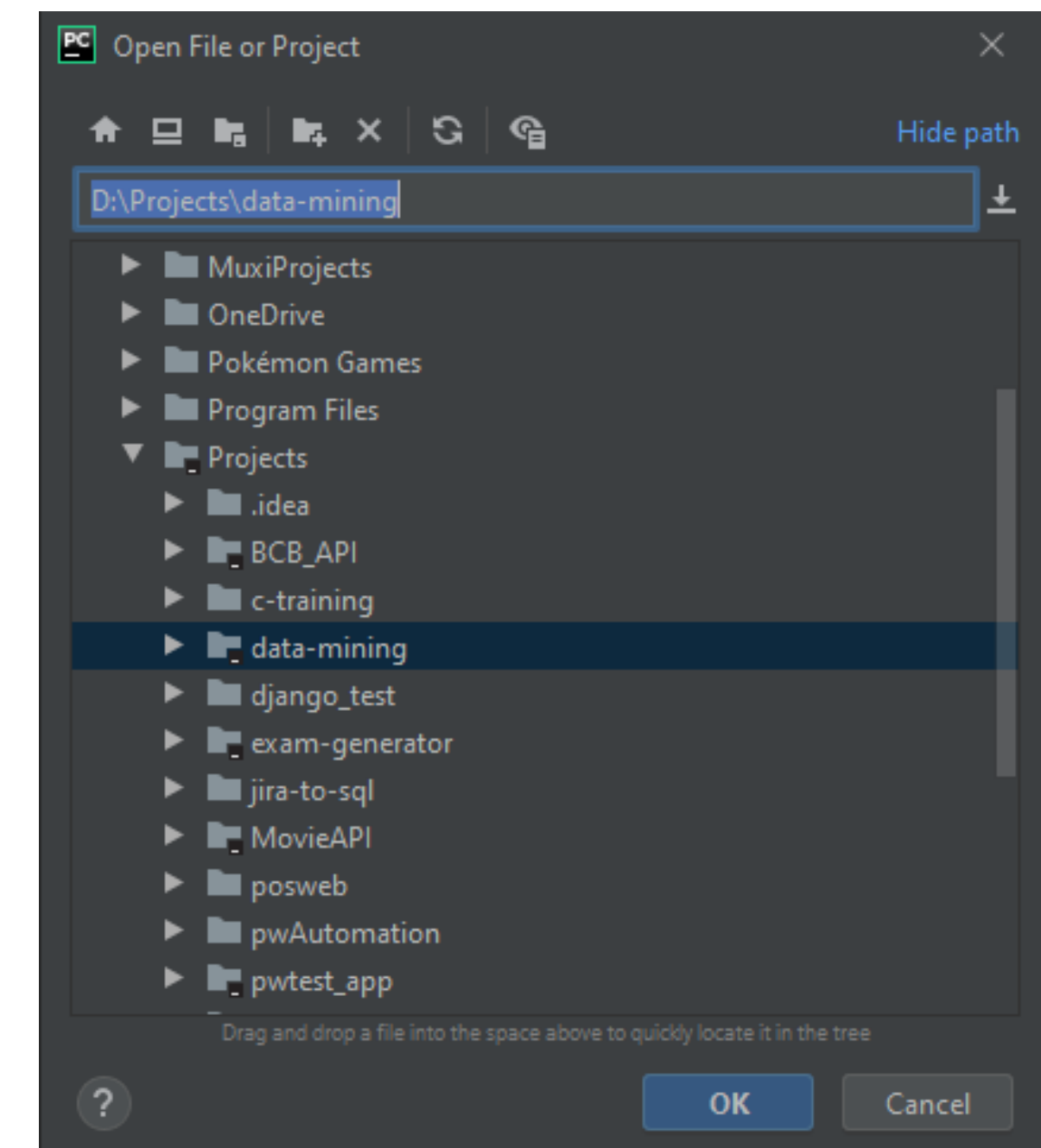
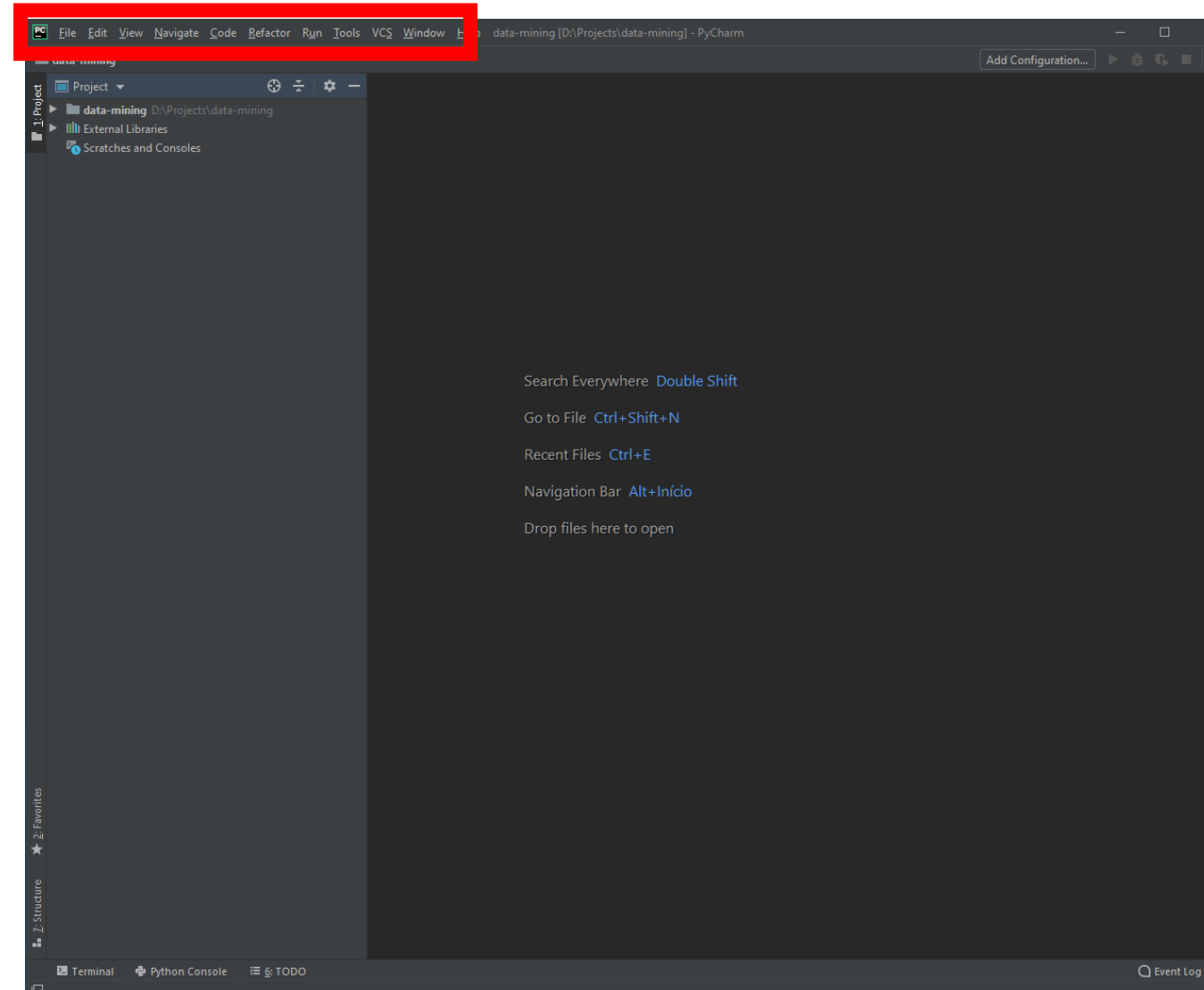
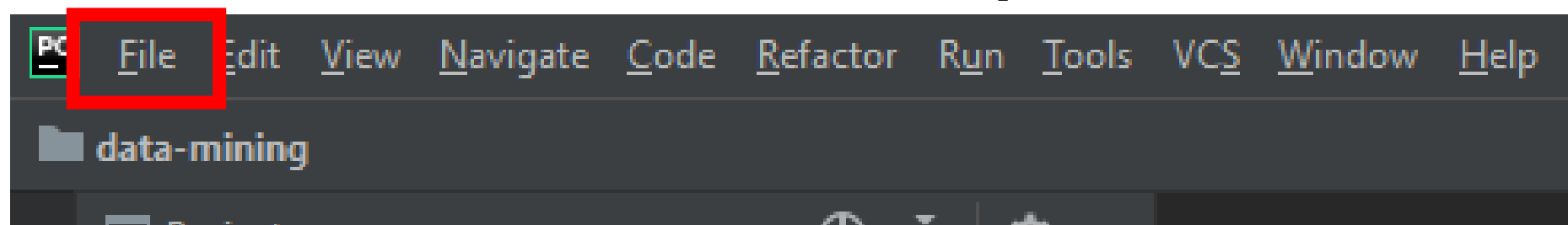
# Iniciando o PyCharm

- Se essa é a primeira vez que abre o PyCharm:
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., D:\Projetos\data-mining);
  - Abra o PyCharm, e na tela inicial clique em **Open**. Selecione a pasta que você criou e clique em **Ok**;



# Iniciando o PyCharm

- Se você já abriu o PyCharm antes:
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., C:\Projetos\data-mining);
  - Abra o PyCharm, e na tela que abrir clique em **File > Open**. Selecione a pasta que você criou e clique em **Ok**;



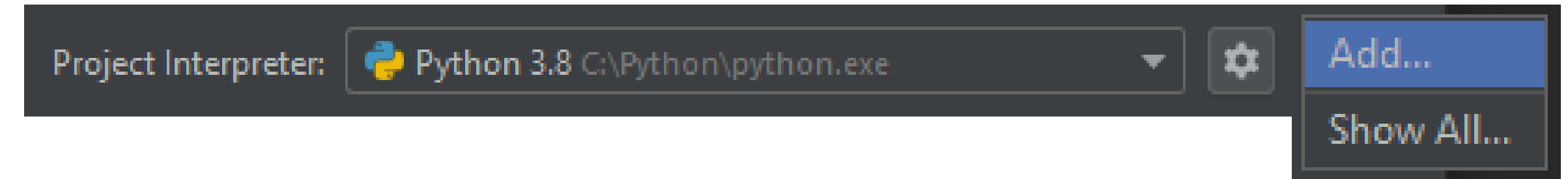


# Algumas configurações do PyCharm

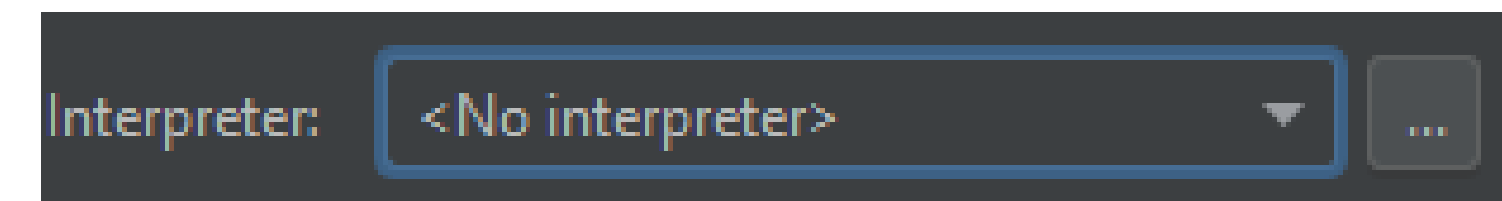
- Mudar o tema para escuro:
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Appearance & Behavior > Appearance**. No campo **Theme**, selecione o tema desejado (minha sugestão é o **Darcula**);
  - Clique em **Ok** para fechar a tela de configurações.

# Algumas configurações do PyCharm

- Configurar o interpretador de Python:



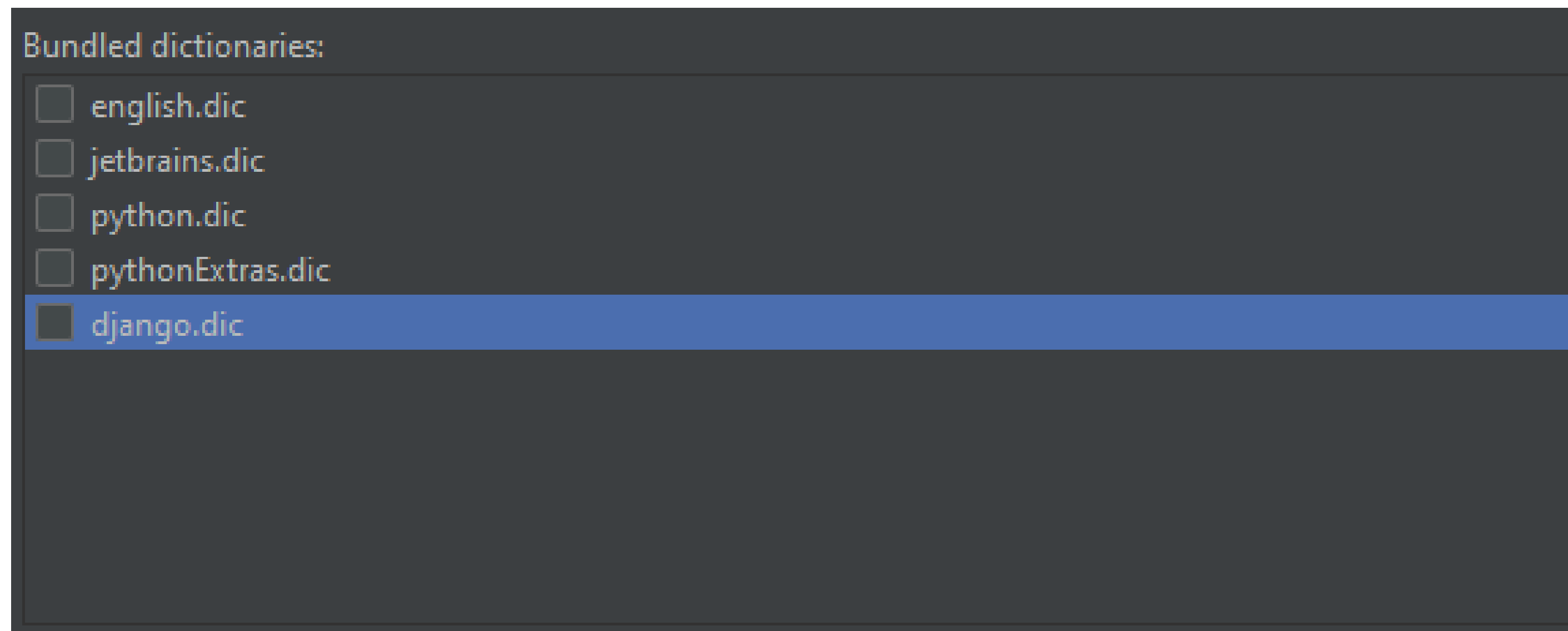
- Clique em **File > Settings**;
- Na janela que aparecer, procure por **Project: <nome-do-projeto> > Project Interpreter**. No campo **Project Interpreter**, verifique se o Python informado é o instalado (no meu caso, **C:\Python\python.exe**). Caso não seja, clique na engrenagem e em **Add...**;
- Na nova janela, clique em **System Interpreter**, e no campo **Interpreter** clique nos três pontos à direita para indicar o local que o seu Python está instalado. Aperte **Ok** até voltar à tela de **Settings**;



- Novamente no campo **Project Interpreter**, altere o Python para refletir o que está instalado. Em seguida clique em **Ok**.

# Algumas configurações do PyCharm

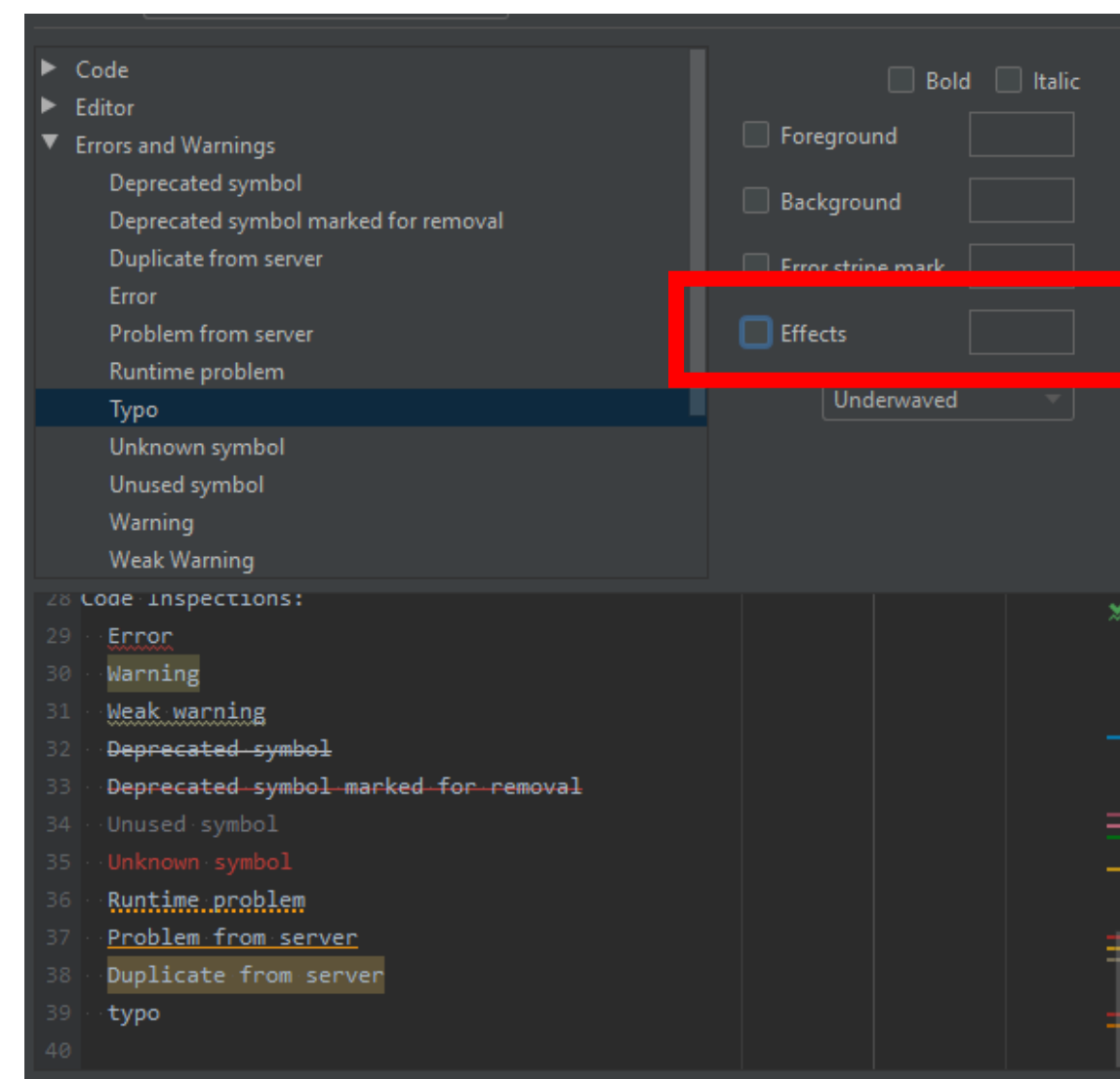
- Desativar inspeção ortográfica:
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Editor > Spelling**. No campo **Bundled dictionaries**, desmarque todas as opções;
  - Clique em **Ok** para sair das configurações.





# Algumas configurações do PyCharm

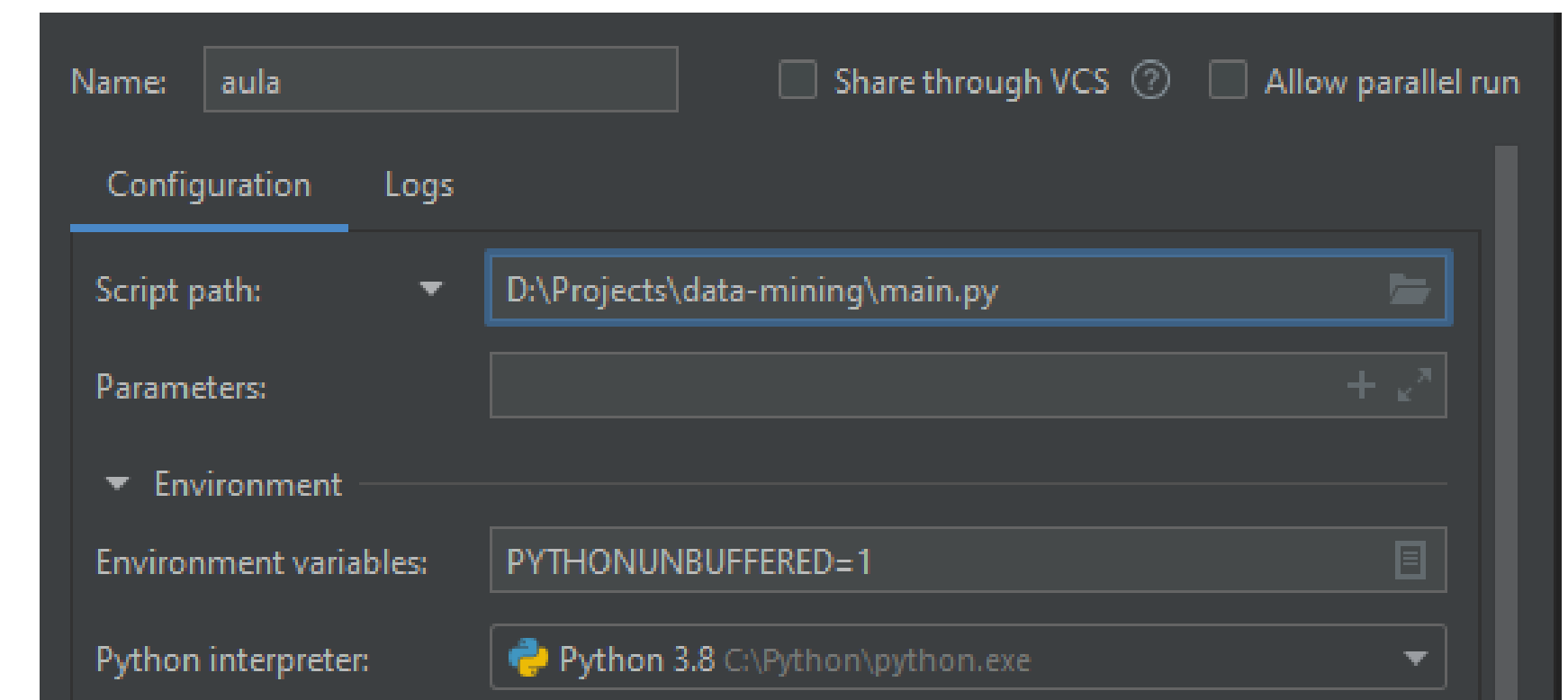
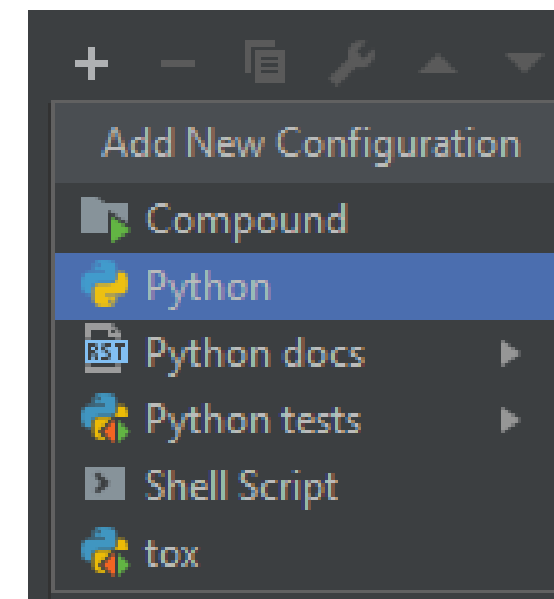
- Desmarcar esquema de cores para *typos*:
  - Clique em **File > Settings**;
  - Na janela que aparecer, procure por **Editor > Color Scheme > General**. No campo **Errors and Warnings > Typo**, desmarque a opção **Effects**;
  - Clique em **Ok** para sair das configurações.





# Algumas configurações do PyCharm

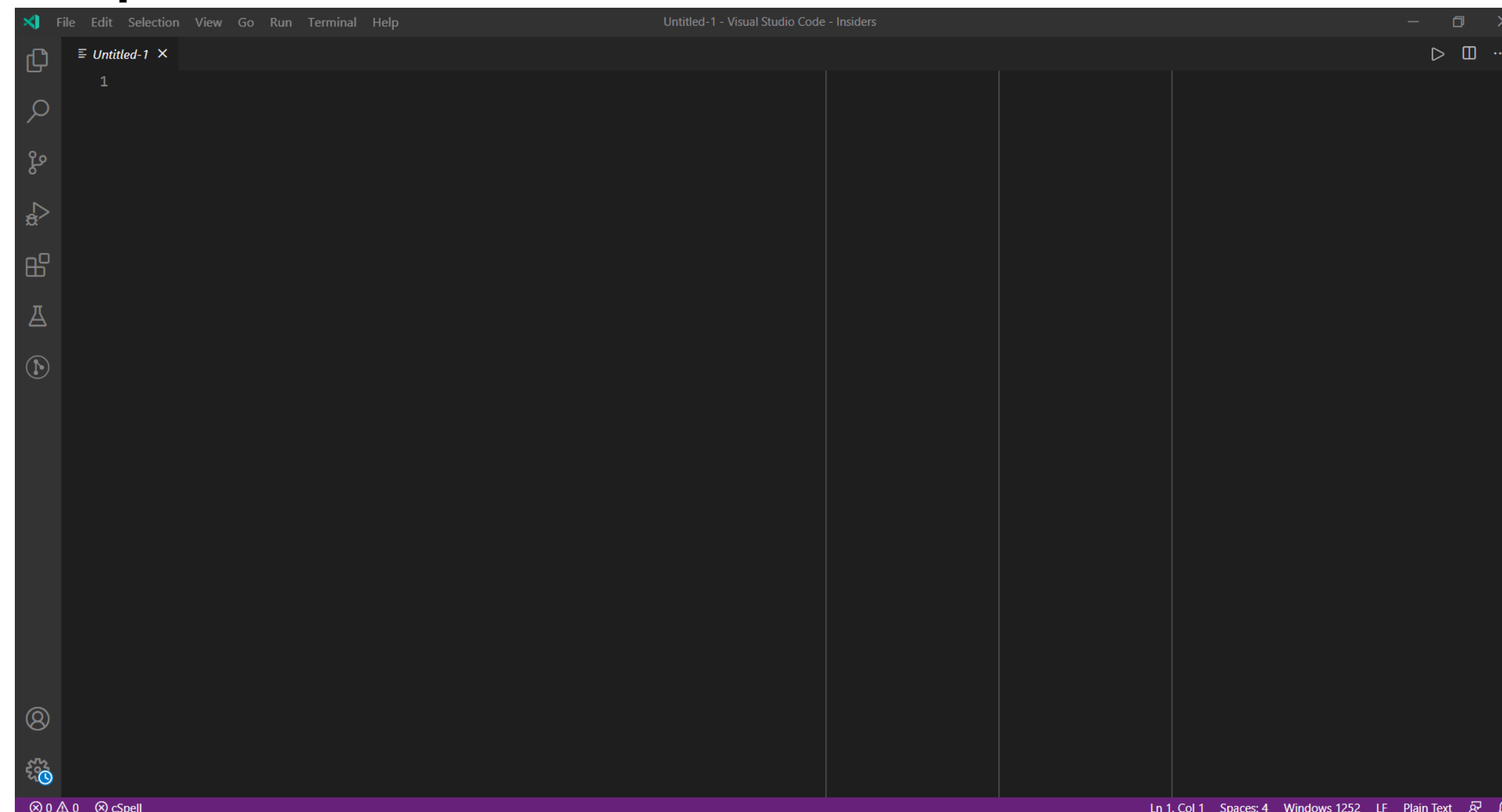
- Configurar uma determinada execução de código:
  - Garanta que as configurações do slide Configurar o interpretador de Python foram executadas;
  - Aperte **Alt + Shift + F10** > **Edit Configurations**, ou na barra de tarefas clique em **Run** > **Edit Configurations**;
  - Na janela que aparecer, no canto superior esquerdo clique no botão de **+** > **Python**;
  - Dê um nome para a execução (p.ex., **aula**) e em **Script path**, informe o caminho do arquivo que você deseja executar;
  - Certifique-se que o **Python interpreter** é o configurado anteriormente, clique em **Apply** e em seguida em **Close**;
  - Para rodar o arquivo, é só usar o atalho **Shift + F10**.



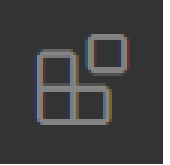
# Utilizando o VSCode

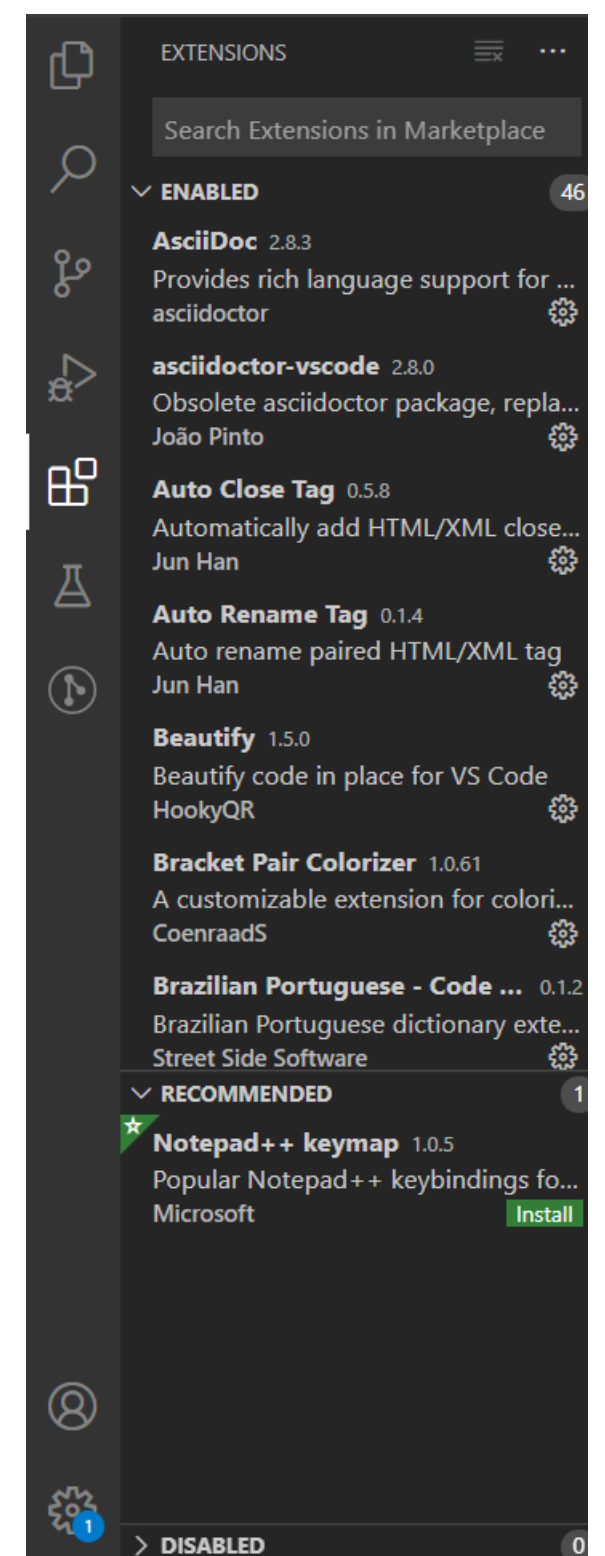
# Iniciando o VSCode

- Se essa é a primeira vez que abre o VSCode:
  - Garanta que você possui uma pasta com o projeto desejado (p.ex., **D:\Projetos\algoritmos**);
  - Abra o VSCode, e na tela inicial clique em **File > Open Folder**. Selecione a pasta que você criou e clique em **Selecionar Pasta**.



# Iniciando o VSCode

- Instalando extensões:
  - Boa parte do atrativo no VSCode é a possibilidade de instalar extensões e customizar a experiência de uso;
  - Para instalar extensões, vá na coluna à esquerda e clique no ícone ;
  - Uma barra de extensões vai aparecer, e você poderá buscar as extensões desejadas e instalá-las;
  - Para o curso, vamos precisar das extensões Python, Beautify, Bracket Pair Colorizer e Visual Studio IntelliCode.





# Iniciando o VSCode

- **Atalhos interessantes no VSCode:**
  - Ctrl + K, Ctrl + O: Abre uma pasta
  - Ctrl + D: Quando uma palavra estiver selecionada, seleciona todas as palavras no arquivo
  - Ctrl + F: Procura por uma palavra ou sentença no arquivo
  - Ctrl + Shift + F: Procura por uma palavra ou sentença em todos os arquivos da pasta
  - Ctrl + H: Substitui uma palavra ou sentença por outra em todo o arquivo
  - Alt + ↓ ou Alt + ↑: Move a linha inteira para baixo ou para cima
  - Shift + Alt + ↓ ou Shift + Alt + ↑: Copia a linha inteira para baixo ou para cima
  - Ctrl + Alt + ↓ ou Ctrl + Alt + ↑: Inclui um ou mais cursores nas linhas abaixo ou acima



# Iniciando o VSCode

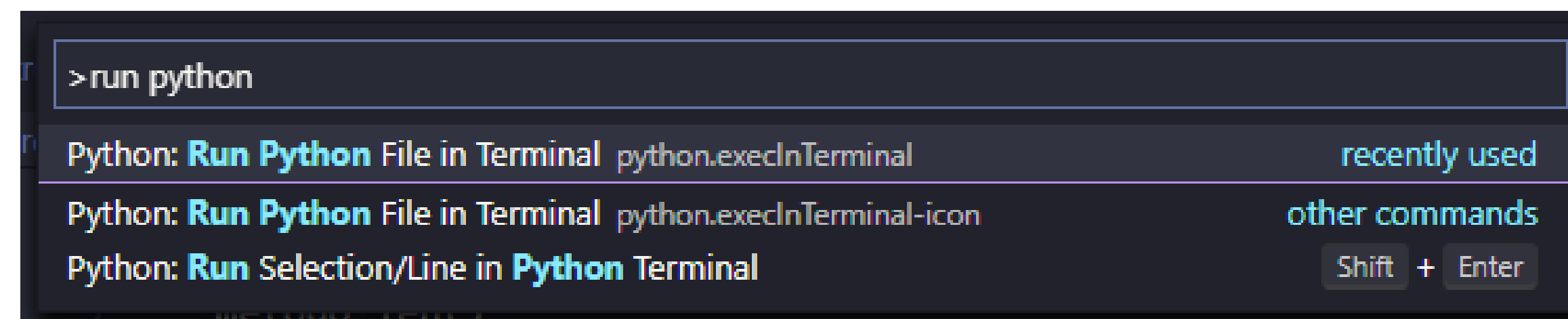
- **Atalhos interessantes no VSCode:**
  - F12: Vai para a declaração de uma variável ou função
  - F12 F12: Mostra todos os usos de uma variável ou função
  - Alt + → ou Alt + ←: Retrocede ou avança na última posição do cursor
  - Ctrl + S: Salva um arquivo
  - Ctrl + P: Abre um arquivo diretamente
  - Ctrl + ,: Abre o menu de configurações
  - Ctrl + Shift + P: Abre uma dropdown com opções de configuração
  - Ctrl + `: Abre a janela do terminal

# Iniciando o VSCode

- **Atalhos interessantes no VSCode:**
  - Ctrl + G: Vai para uma linha específica do arquivo
  - Ctrl + ]: Divide a tela em duas
  - Ctrl + 1 (ou 2, 3, 4): Seleciona um painel específico
  - Alt + Shift + O: Alterna entre divisão na horizontal ou na vertical
  - Ctrl + F4 ou Ctrl + W: Fecha o arquivo selecionado (se for um painel selecionado e ele estiver vazio, fecha o painel)
  - Ctrl + ;: Comenta a(s) linha(s) selecionada(s)

# Iniciando o VSCode

- Executando um código Python no VSCode:
  - Aperte as teclas Ctrl + Shift + P;
  - Na caixa de busca que aparece, digite “Run Python File in Terminal”, selecione a opção adequada e aperte **Enter**;
  - O VSCode deve rodar o código no terminal do próprio IDE.





# Configurando o pylint

# Configurando o pylint

- O pylint é um pacote do Python que auxilia na adequação do código aos padrões de programação da comunidade Python
- Para instalar o pylint...
  - ...para MacOS veja [aqui](#)
  - ...para Windows veja [aqui](#)
- Para executar o pylint basta entrar com o comando **pylint <caminho>**, com o caminho do arquivo que deseja rodar o linter.

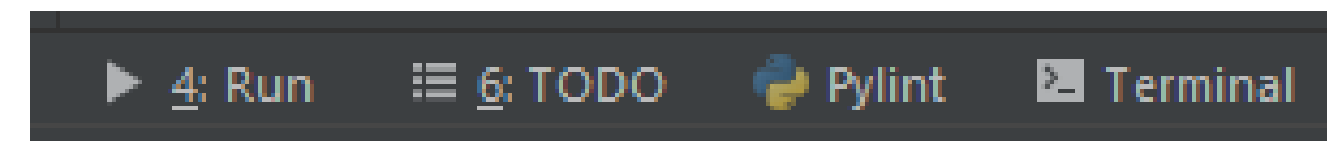
```
C:\Users\vmachado>pylint D:\Victor\Pessoal\IBMEC\2020.2\ALG\Aulas\teste.py
***** Module teste
D:\Victor\Pessoal\IBMEC\2020.2\ALG\Aulas\teste.py:1:0: C0114: Missing module docstring (missing-module-docstring)

-----
Your code has been rated at 0.00/10
```



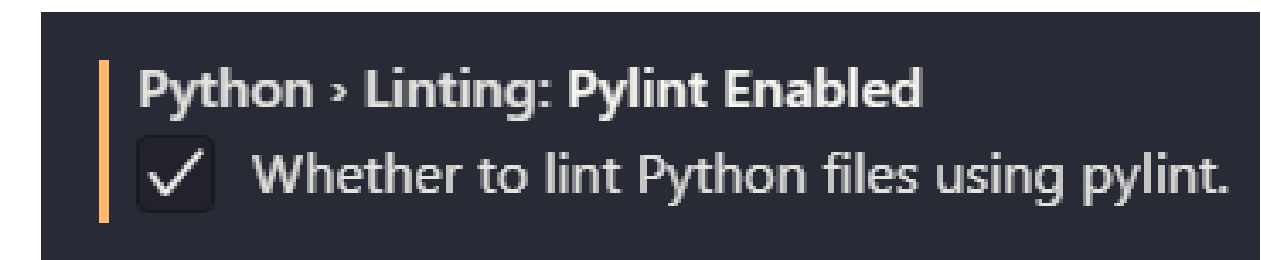
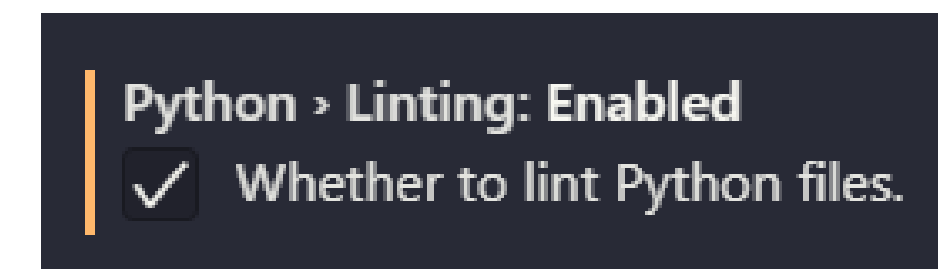
# Configurando o pylint

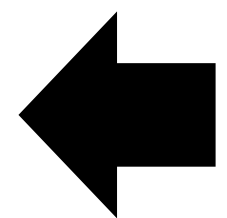
- O pylint ainda pode ser rodado direto no terminal do VSCode ou do PyCharm, basta abrir o terminal e seguir os mesmos passos mencionados anteriormente. No entanto, os ambos os IDEs fornecem meios de utilizar o pylint diretamente durante a implementação do código.
- **Usando o pylint no PyCharm:**
  - O PyCharm já possui a inspeção automática do editor, após configurar o interpretador. Para identificar as inspeções vá em **File > Settings** e na janela clique em **Editor** e depois em **Inspections**;
  - O PyCharm também possui um plugin. Em **File > Settings**, clique em **Plugins** e busque por **Pylint**. Instale e reinicie o IDE. Sempre que abrir um arquivo Python o IDE vai incluir a opção “Pylint” no canto inferior esquerdo. Para executar basta clicar na opção e depois em “Run”.



# Configurando o pylint

- Usando o pylint no VSCode:
  - Com a extensão “Python” instalada no VSCode e o pacote pylint instalado, abra o menu de configurações (use o atalho **Ctrl + ,** ou vá em **File > Preferences > Settings**) e procure por python linting;
  - Marque as seguintes opções:
    - Python > Linting: Enabled
    - Python > Linting: Lint On Save
    - Python > Linting: Pylint Enabled
  - Para rodar o pylint sem ser pelo terminal, use o atalho **Ctrl + Shift + P** e em seguida digite “Run Linting” e aperte Enter. O IDE vai marcar no código os problemas.






# Instalando pacotes pelo PyPI

# Instalando pacotes via PyPI

- Uma das grandes vantagens ao se programar em Python é ter à disposição uma gama de pacotes e bibliotecas disponíveis pela própria comunidade, que desenvolve novas funcionalidades e distribui online, na maioria das vezes de forma gratuita.
- Um dos locais mais confiáveis e mais simples de se obter um novo pacote é através do PyPI, um repositório oficial de pacotes da linguagem, mantido pela própria organização que mantém o Python.
- O PyPI é acessado utilizando uma ferramenta chamada **pip**, que é instalada automaticamente ao se instalar o interpretador de Python. Portanto, a instalação de novos pacotes é muito fácil de se realizar, com apenas uma linha de comando.

# Instalando pacotes pelo PyPI

- Instalando pacotes via pip no Windows:

- Clique no botão do Windows  ;
- Digite a caixa de pesquisa **prompt de comando**, e abra o programa;
- No terminal entre com o seguinte comando:

```
C:\Users\vmachado>C:\Python\python.exe -m pip install pylint
```

- Lembre-se de substituir o caminho do Python para o caminho instalado no seu computador, e altere **pylint** para o pacote escolhido.



# Instalando pacotes pelo PyPI

- A instalação do Python no MacOS não costuma vir com o pip. Caso não tenha vindo, tente fazer os passos abaixo primeiro:

- Abra o terminal (pasta **Applications > Utilities > Terminal**);
- Entre com os seguintes comandos:

```
curl https://bootstrap.pypa.io/get-pip.py > get-pip.py  
sudo python get-pip.py
```

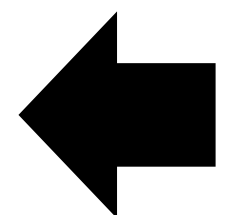
- Para instalar um pacote via pip no MacOS:

- Usando o mesmo terminal usado para instalar o pip, entre com o seguinte comando:

```
sudo pip install <nome_do_pacote>
```

- Veja o vídeo abaixo para mais detalhes:

- <https://www.youtube.com/watch?v=yBdZZGPpYxg>



# Noções básicas de OO

# Por que usar OO?

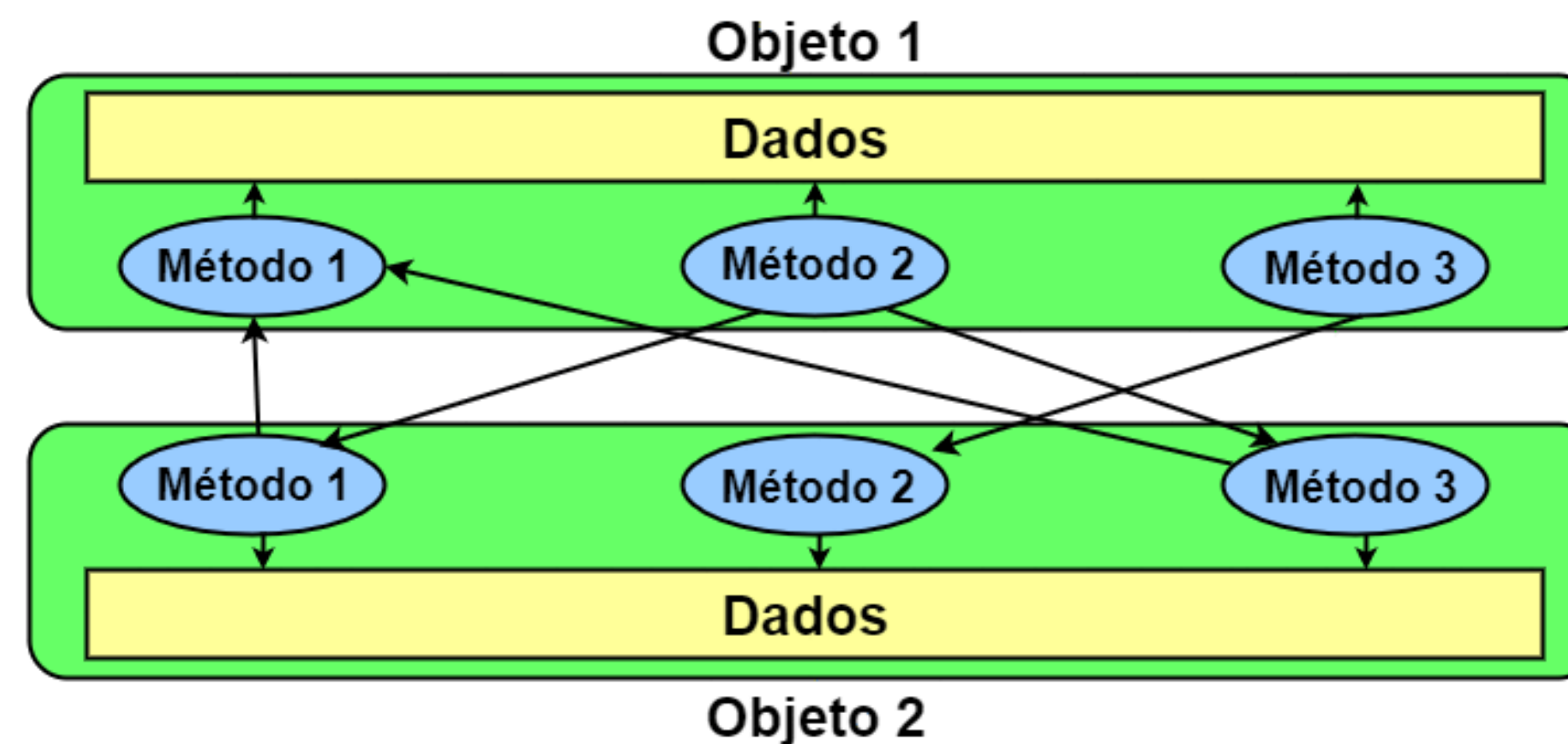
- Segundo o Paradigma Procedural, é possível representar todo e qualquer processo do mundo real a partir da utilização de **apenas** três estruturas básicas:
  - Sequência: Os passos devem ser executados um após o outro, linearmente. Ou seja, o programa seria uma sequência finita de passos. Em uma unidade de código, todos os passos devem ser feitos para se programar o algoritmo desejado;
  - Decisão: Uma determinada sequência de código pode ou não ser executada. Para isto, um teste lógico deve ser realizado para determinar ou não sua execução. A partir disto, verifica-se que duas estruturas de decisão (também conhecida como seleção) podem ser usadas: a if-else e a switch.
  - Iteração: É a execução repetitiva de um segmento (parte do programa). A partir da execução de um teste lógico, a repetição é realizada um número finito de vezes. Estruturas de repetição conhecidas são: for, foreach, while, do-while, repeat-until, entre outras (dependendo da linguagem de programação).

# Por que usar OO?

- Usar apenas essas três estruturas pode apresentar algumas limitações:
  - Quanto mais complexo o programa se torna, mais difícil fica a manutenção de uma sequência organizada de código;
  - Com esse paradigma é muito fácil deixar o código extenso e com muitas duplicações;
  - Mesmo modularizações providas pelas linguagens podem deixar o código muito complexo.
- Em resumo, a simplificação da representação das reais necessidades dos problemas a serem automatizados leva a uma facilidade de entendimento e representação. Porém, isso pode levar a uma complexidade de programação caso o nicho de negócio do sistema-alvo seja complexo.

# Por que usar OO?

- Ao contrário do paradigma procedural, a OO preconiza que os dados relativos a uma representação de uma entidade do mundo real devem somente estar juntos de suas operações, quais são os responsáveis por manipular - exclusivamente - tais dados.
- Assim, há uma separação de dados e operações que não dizem respeito a uma mesma entidade. Todavia, se tais entidades necessitarem trocar informações, farão isto através da chamada de seus métodos, e não de acessos diretos a informações da outra.





# Fundamentos da OO

- Abstração:
  - Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos.
  - Não devemos nos preocupar com características menos importantes, ou seja, acidentais. Devemos, neste caso, nos concentrar apenas nos aspectos essenciais. Por natureza, as abstrações devem ser incompletas e imprecisas.



# Fundamentos da OO

- Abstração:
  - Com a abstração dos conceitos, conseguimos reaproveitar, de forma mais eficiente, o nosso “molde” inicial, que pode ser detalhado conforme for necessário para o nosso caso específico.
  - Os processos de inicialmente se pensar no mais abstrato e, posteriormente, acrescentar ou se adaptar são também conhecidos como **generalização** e **especialização**, respectivamente.



# Fundamentos da OO

- Reuso:
  - Não existe pior prática em programação do que a repetição de código. Isto leva a um código frágil, propício a resultados inesperados. Quanto mais códigos são repetidos pela aplicação, mais difícil vai se tornando sua manutenção.
  - O fato de simplesmente utilizarmos uma linguagem OO não é suficiente para se atingir a reusabilidade, temos de trabalhar de forma eficiente para aplicar os conceitos de **herança e associação**, por exemplo.
  - Na herança, é possível criar classes a partir de outras classes. A classe filha, além do que já foi reaproveitada, pode acrescentar o que for necessário para si.
  - Já na associação, o reaproveitamento é diferente. Uma classe pede ajuda a outra para poder fazer o que ela não consegue fazer por si só. Em vez de simplesmente repetir, em si, o código que está em outra classe, a associação permite que uma classe forneça uma porção de código a outra. Assim, esta troca mútua culmina por evitar a repetição de código.

# Fundamentos da OO

- Encapsulamento:
  - Quando alguém se consulta com um médico, por estar com um resfriado, seria desesperador se ao final da consulta o médico entregasse a seguinte receita:

Receituário (Complexo)

- 400mg de ácido acetilsalicílico
- 1mg de maleato de dexclorfeniramina
- 10mg de cloridrato de fenilefrina
- 30mg de cafeína

Misturar bem e ingerir com água. Repetir em momentos de crise.

- A primeira coisa que viria em mente seria: onde achar essas substâncias? Será que é vendido tão pouco? Como misturá-las? Existe alguma sequência? Seria uma tarefa difícil – até complexa – de ser realizada. Mais simples do que isso é o que os médicos realmente fazer: passam uma cápsula onde todas estas substâncias já estão prontas. Ou seja, elas já vêm encapsuladas.



# Fundamentos da OO

- Encapsulamento:
  - Com isso, não será preciso se preocupar em saber quanto e como as substâncias devem ser manipuladas para no final termos o comprimido que resolverá o problema. O que interessa é o resultado final, no caso, a cura do resfriado. A complexidade de chegar a essas medidas e como misturá-las não interessa. É um processo que não precisa ser do conhecimento do paciente.

Receituário (Encapsulado)

1 comprimido de Resfriol. Ingerir com água. Repetir em momentos de crise.

- Essa mesma ideia se aplica na OO. No caso, a complexidade que desejamos esconder é a de implementação de alguma necessidade. Com o encapsulamento, podemos esconder a forma como algo foi feito, dando a quem precisa apenas o resultado gerado.
- Uma vantagem deste princípio é que as mudanças se tornam transparentes, ou seja, quem usa algum processamento não será afetado quando seu comportamento interno mudar.

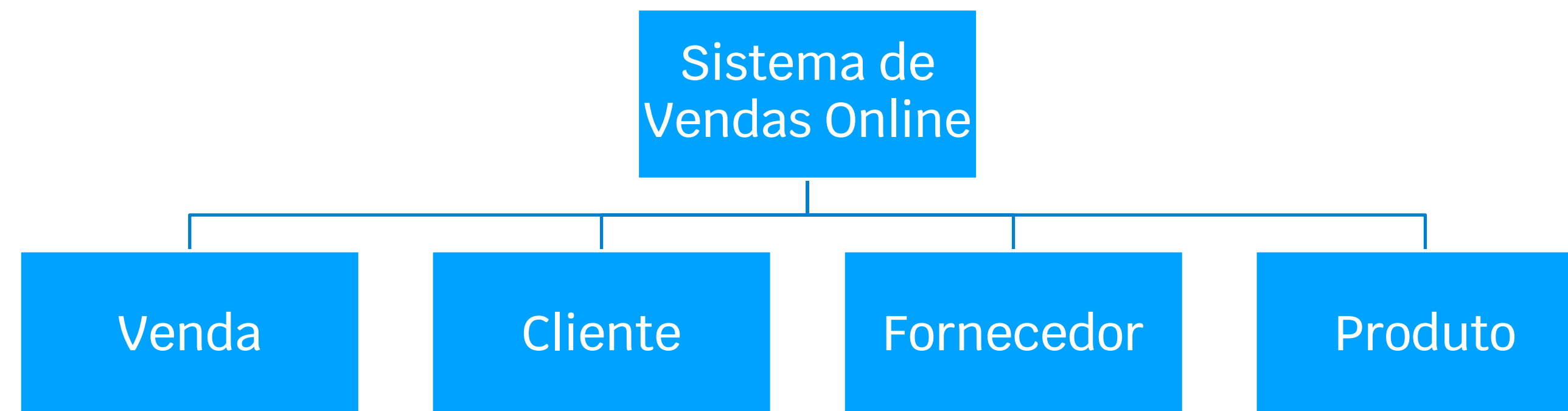
# Conceitos estruturais em OO

- Embora a OO tenha vantagens em relação aos paradigmas que a precederam, existe uma desvantagem inicial: ser um modo mais complexo e difícil de se pensar. Isso pode ser atribuído à grande quantidade de conceitos que devem ser assimilados para podermos trabalhar orientado a objetos.
- Vamos falar de quatro conceitos estruturais principais:
  - A classe
  - O atributo
  - O método
  - O objeto



# Classes

- Apesar do paradigma ser nomeado como Orientado a Objetos, tudo começa com a definição de uma classe.
- Antes mesmo de ser possível manipular objetos, é preciso definir uma classe, pois esta é a unidade inicial e mínima de código na OO. É a partir de classes que futuramente será possível criar objetos.
- O objetivo de uma classe é definir, servir de base, para o que futuramente será o objeto.
  - A classe é um “molde” que deverá ser seguido pelos objetos.
- Uma classe pode ser definida como uma abstração de uma entidade, seja ela física (bola, pessoa, carro, etc.) ou conceitual (viagem, venda, estoque, etc.) do mundo real.



# Classes

- O nome de uma classe deve representar bem sua finalidade dentro do contexto ao qual ela foi necessária.
  - Em um sistema de controle hospitalar, podemos ter uma classe chamada **Pessoa**;
  - Em um sistema de ponto de vendas (PDV), também temos o conceito de **Pessoa**.
- Entretanto, nota-se que o termo *pessoa* pode gerar uma ambiguidade, embora esteja correto.
- Assim, recomenda-se ser mais específico na nomeação das classes, como **Médico**, **Paciente**, **Cliente**, **Vendedor**.
- Embora possa parecer preciosismo, classes com nomes pobremente definidos podem dificultar o entendimento do código e até levar a erros de utilização. Pense bem antes de nomear uma classe.

# Atributos

- Após o processo inicial de identificar as entidades (classes) que devem ser manipuladas, começa a surgir a necessidade de detalhá-las.
  - Quais informações devem ser manipuladas através desta classe?
- Devemos, então, caracterizar as classes definidas. Essas características é que vão definir quais informações as classes poderão armazenar e manipular. Na OO, estas características e informações são denominadas de **atributo**.
- Essa definição deixa bem claro que os atributos devem ser definidos dentro da classe. É a partir do uso de atributos que será possível caracterizar (detalhar) as classes.
- Assim como nas classes, os atributos podem ser representados a partir de substantivos. Além destes, podemos também usar adjetivos. Pensar em ambos pode facilitar o processo de identificação dos atributos.

# Atributos

- Classe **Paciente** em um sistema hospitalar → quais seriam os possíveis atributos?
  - Nome
  - CPF
  - Data de nascimento
  - Histórico médico
- Todos estes são substantivos, mas alguns de seus valores poderiam ser adjetivos.
- Quanto mais for realizado o processo de caracterização, mais detalhada será a classe e, com isso, ela terá mais atributos.
- Porém, é preciso ter parcimônia no processo de identificação dos atributos!
  - Hobby?
  - Possui carro?

# Atributos

- Nem sempre uma informação, mesmo sendo importante, deve ser transformada em um atributo. Por exemplo, **idade**. Ter que recalcular a idade toda vez que a pessoa fizer aniversário é custoso e propenso a erros.
- Neste caso, seria melhor usar o que é conhecido como **atributo calculado** ou **atributo derivado**. A idade não se torna um atributo em si, mas tem seu valor obtido a partir de um método.
- Não diferentemente de linguagens estruturadas, um atributo possui um tipo. Como sua finalidade é armazenar um valor que será usado para caracterizar a classe, ele precisará identificar qual o tipo do valor armazenado em si. Linguagens orientadas a objetos proveem os mesmos tipos de dados básicos – com pequenas variações – que suas antecessoras.



# Atributos

- A nomeação de atributos deve seguir a mesma preocupação das classes: deve ser o mais representativo possível. Nomes como **qtd**, **vlr** devem ser evitados.
- Esses nomes eram válidos na época em que as linguagens de programação e os computadores que as executavam eram limitados, portanto deveríamos sempre abreviar os nomes das variáveis.
- Entretanto, atualmente não temos essa limitação, e os editores modernos possuem recursos para reaproveitar nomes de atributos e variáveis sem precisar digitar tudo novamente. Portanto, passe a usar **quantidade** e **valor**.
- Utilize nomes claros. Evite, por exemplo, o atributo **data**. Uma data pode significar várias coisas: nascimento, morte, envio de um produto, cancelamento de venda. Escreva exatamente o que se deseja armazenar nesse atributo: **data\_nascimento**, **data\_obito**, etc.



# Métodos

- Tendo identificado a classe com seus atributos, as seguintes perguntas podem surgir:
  - Mas o que fazer com eles?
  - Como utilizar a classe e manipular os atributos?
- É nessa hora que o método entra em cena. Este é responsável por identificar e executar as operações que a classe fornecerá. Essas operações, via de regra, têm como finalidade manipular os atributos.
- Para facilitar o processo de identificação dos métodos de uma classe, podemos pensar em verbos. Isso ocorre devido à sua própria definição: **ações**. Ou seja, quando se pensa nas ações que uma classe venha a oferecer, estas identificam seus métodos.

# Métodos

- No processo de definição de um método, a sua assinatura deve ser identificada. Esta nada mais é do que o nome do método e sua lista de parâmetros. Mas como nomear os métodos? Novamente, uma expressividade ao nome do método deve ser fornecida, assim como foi feito com o atributo.
- Por exemplo, no contexto do hospital, imagine termos uma classe **Procedimento**, logo, um péssimo nome de método seria **calcular**. Calcular o quê? O valor total do procedimento, o quanto cada médico deve receber por ele, o lucro do plano? Neste caso, seria mais interessante **calcular\_total**, **calcular\_ganhos\_medico**, **calcular\_lucro**.
- Veja que, ao lermos esses nomes, logo de cara já sabemos o que cada método se propõe a fazer. Já a lista de parâmetros são informações auxiliares que podem ser passadas aos métodos para que estes executem suas ações. Cada método terá sua lista específica, caso haja necessidade. Esta é bem livre e, em determinados momentos, podemos não ter parâmetros, como em outros podemos ter uma classe passada como parâmetro, ou também tipos primitivos e classes ao mesmo tempo.

# Métodos

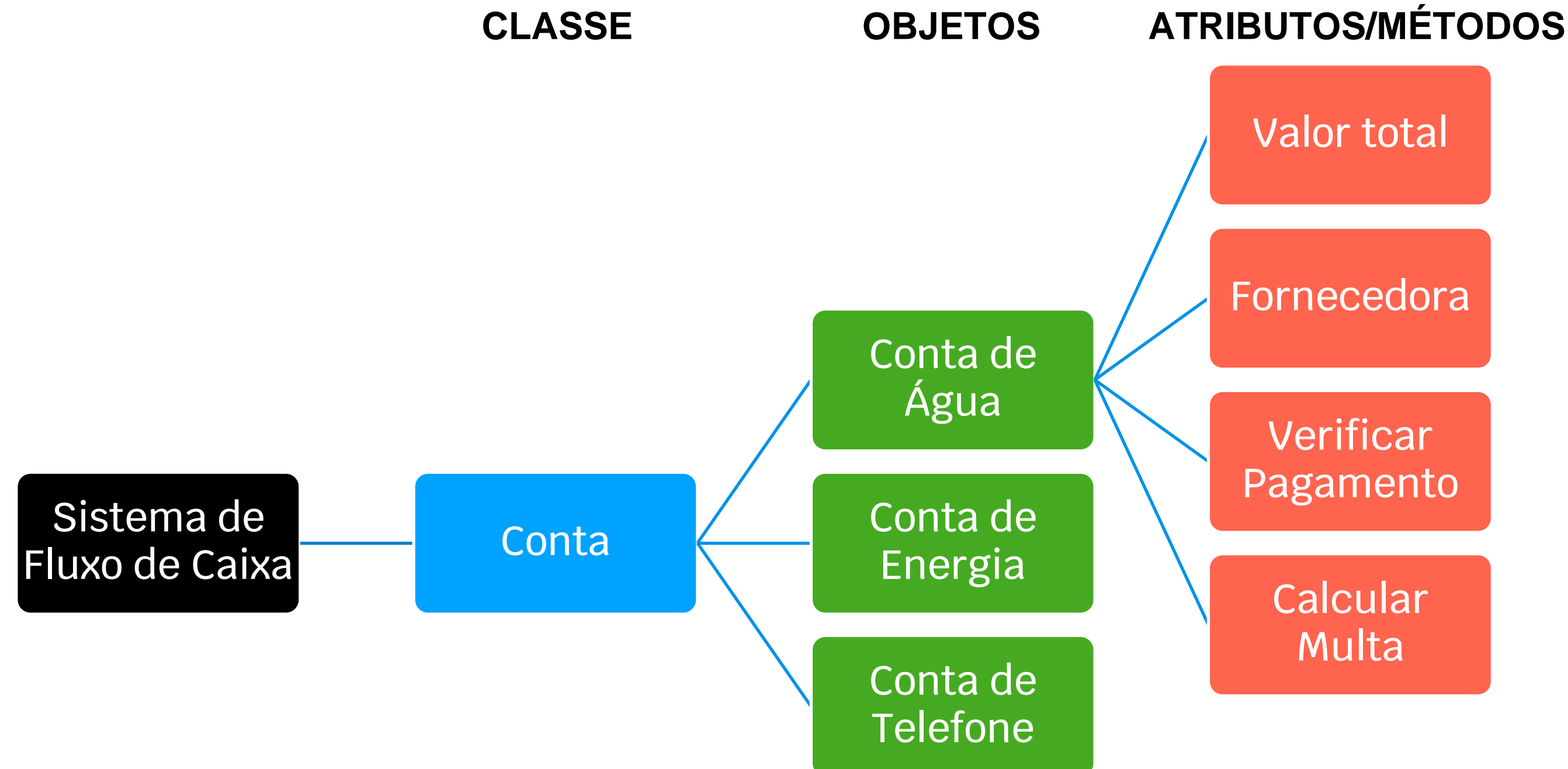
- Há também a possibilidade de passarmos somente tipos primitivos, entretanto, isto remete à programação procedural e deve ser desencorajado. Via de regra, se você passa muitos parâmetros separados, talvez eles pudessem representar algum conceito em conjunto. Neste caso, valeria a pena avaliarmos se não seria melhor criar uma classe para aglutiná-los.
- Por fim, embora não faça parte de sua assinatura, os métodos devem possuir um retorno. Se uma ação é disparada, é de se esperar uma reação. O retorno de um método pode ser qualquer um dos tipos primitivos vistos na seção sobre atributos.
- Além destes, o método pode também retornar qualquer um dos conceitos (classes) que foram definidos para satisfazer as necessidades do sistema em desenvolvimento, ou também qualquer outra classe - não criada pelo programador - que pertença à linguagem de programação escolhida.

# O método construtor

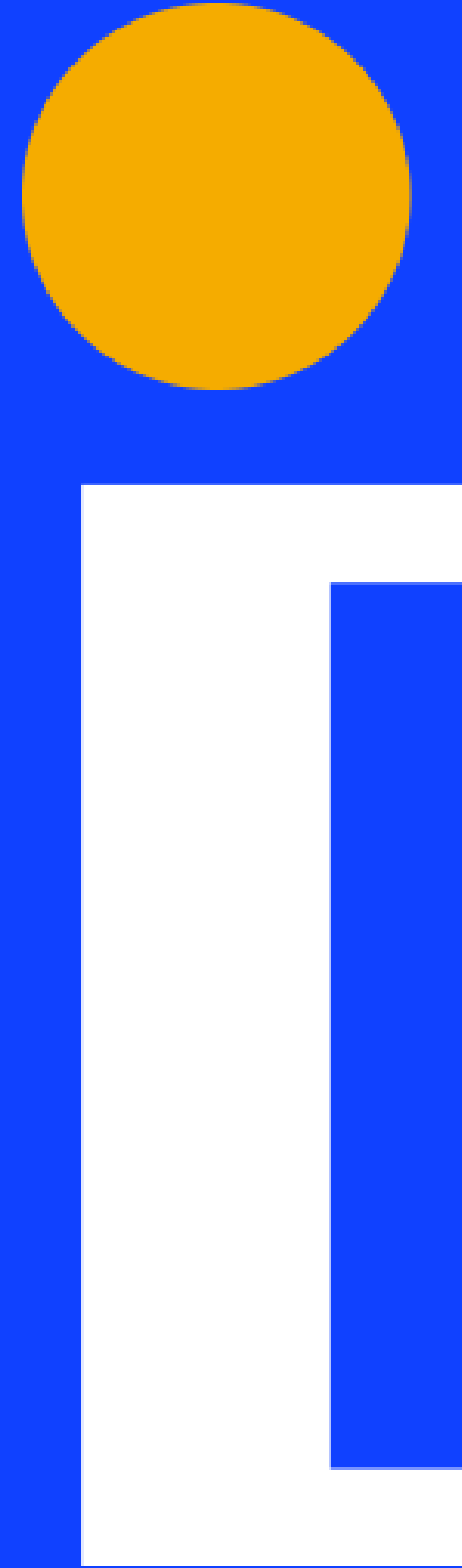
- Em uma classe, independente de qual conceito ela queira representar, podemos ter quantos métodos forem necessários. Cada um será responsável por uma determinada operação que a classe deseja oferecer. Além disso, independente da quantidade e da finalidade dos métodos de uma classe, toda classe possui um método chamado **construtor**.
- Características do construtor:
  - Responsável por criar objetos a partir da classe em questão;
  - Prover valores iniciais para o objeto;
  - Possui nome igual ao da classe;

# Objetos

- Objeto é a instanciação de uma classe.
- Como já explicado, a classe é a abstração base a partir da qual os objetos serão criados. Quando se usa a OO para criar um software, primeiro pensamos nos objetos que ele vai manipular/representar. Tendo estes sido identificados, devemos então definir as classes que servirão de abstração base para que os objetos venham a ser criados (instanciados).







IBMEC.BR

 /IBMEC

 IBMEC

 @IBMEC\_OFICIAL

 @IBMEC

 **ibmec**