

Programação

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



Índice

Introdução ao curso

- Afinal, o que é programar?
- Linguagens de programação
- Sobre o Python
- Configurando o ambiente Python
- Instalando e configurando IDEs
 - Utilizando o PyCharm
 - Utilizando o VSCode
- Instalando pacotes pelo PyPI
 - Configurando o pylint

Índice

Algoritmos e Lógica de Programação

- O que são algoritmos?
- Como representar algoritmos
- Estruturas de seleção
- Estruturas de repetição

Noções básicas de OO

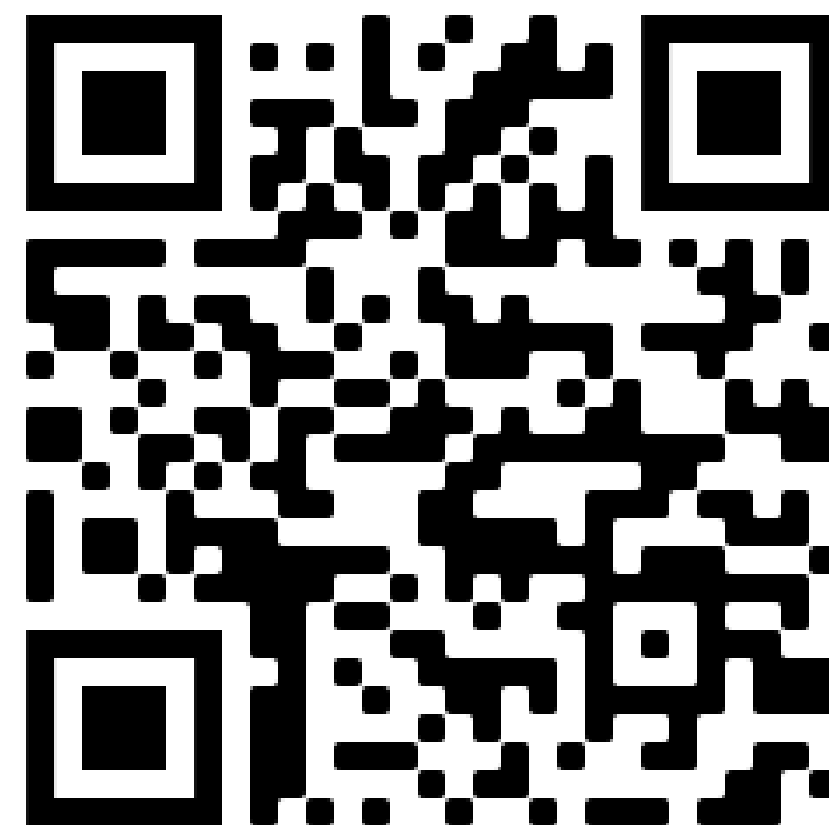
Git

Introdução ao curso

Apresentação do curso

Informações gerais:

- Contato: victor.silva@professores.ibmec.edu.br
- Aulas às quintas-feiras, de 18:40 às 22:30
- Grupo no Whatsapp: <https://chat.whatsapp.com/IxoNQcCEhBICyBq0Ls53en>
- Site com materiais: <http://victor0machado.github.io>



Apresentação do curso

Avaliação

- Proporção:
 - AC (20%): Atividades em sala, individuais ou em dupla
 - AP1 e AP2 (40% cada): Projetos de disciplina, em grupo; avaliação teórica, individual
- AS será uma prova individual, que substituirá a menor nota entre AP1 e AP2.
- Para AP1 e AP2, grupos de no mínimo 2 e no máximo 3 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>

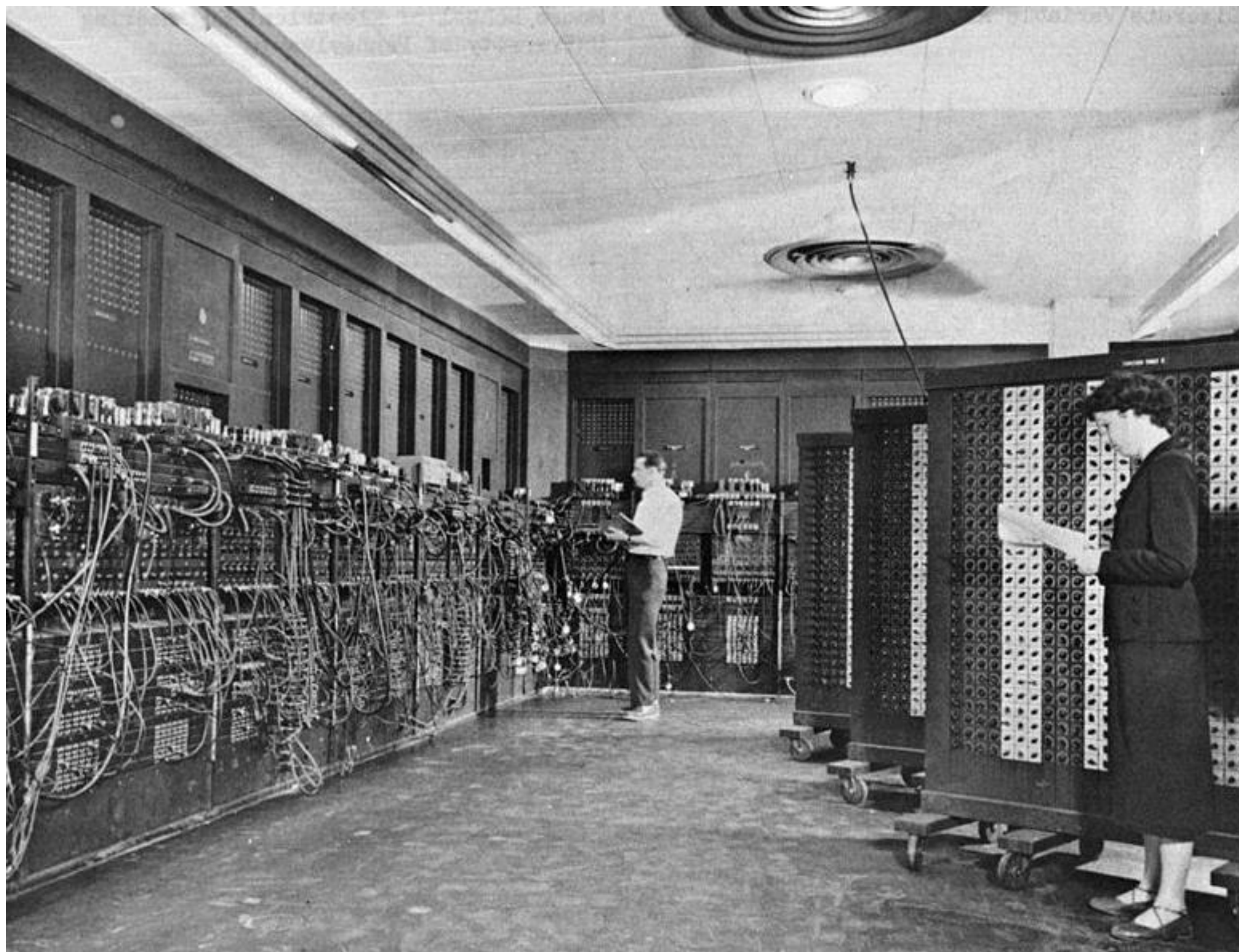
Sugestões de materiais para estudo

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>

Afinal, o que é “programar”?

Um pouco de história...



- ENIAC, primeiro computador eletrônico da história, entrou em operação logo após a II Guerra Mundial
- 30 toneladas, em uma sala de 10m x 15m
- Dezenas de pessoas operando ao mesmo tempo, como telefonistas

Programação era configurar o computador para que ele realize operações específicas

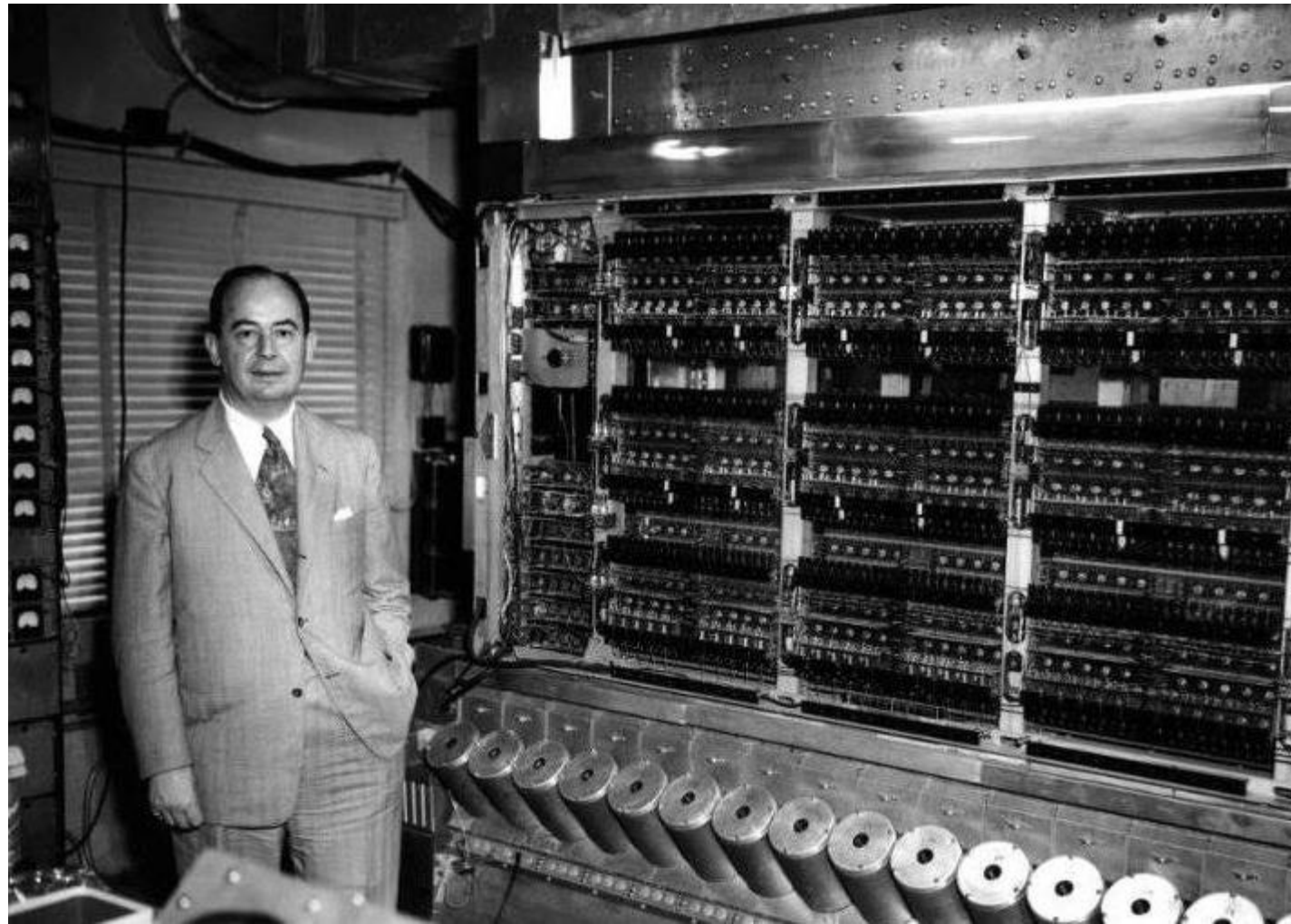
Afinal, o que é “programar”?

Limitações da época:

- Computadores com programas fixos
 - Uma alteração simples no programa do ENIAC poderia demorar três semanas ou mais para que ele voltasse a trabalhar!
 - Computadores eram como calculadoras de mesa...
- Processo de programação era altamente complexo, envolvia configurar sinais específicos em linguagem binária, ou seja, utilizando 0s e 1s
- Os computadores ficavam restritos a grandes corporações ou a instituições governamentais

Afinal, o que é “programar”?

Duas pessoas fundamentais para a mudança de paradigma



John von Neumann, 1903-1957

- Um dos construtores do ENIAC
- Desenvolveu a **arquitetura de von Neumann**, usada até hoje
- Permitiu a construção de computadores de programas armazenados



Grace Hopper, 1906-1992

- Uma das construtoras do UNIVAC I
- Criadora do primeiro compilador da história, o A-0
- Desenvolveu uma das primeiras linguagens de programação de alto nível, o Flow-Matic
- Possível criadora do termo “bug” para indicar defeitos no software

Afinal, o que é “programar”?

Com a criação dos compiladores e a constante evolução dos hardwares e softwares, o processo de programação se tornou muito mais simples.

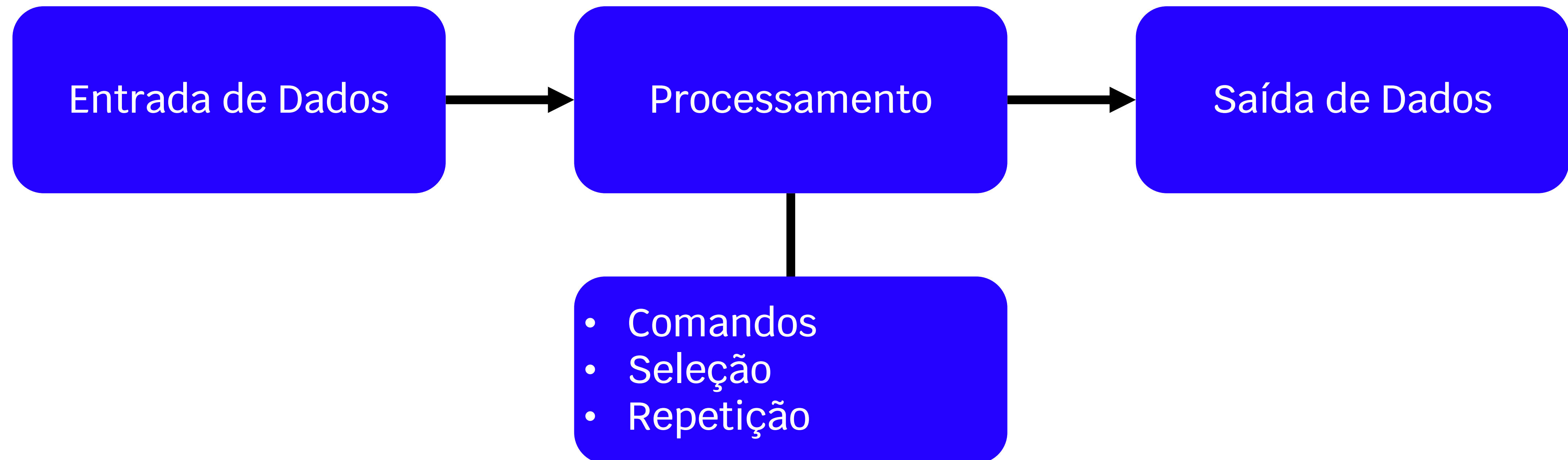
Em essência, sua definição continua a mesma:

Programar um computador significa fornecer à máquina um conjunto de instruções que indica o que você quer que ela 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.

Afinal, o que é “programar”?

A programação se resume a uns poucos conceitos fundamentais



Linguagens de programação

O conceito de compiladores criado por Grace Hopper trouxe uma nova perspectiva sobre como programar. A partir de então, já não era mais necessário escrever código em linguagem de máquina, ou seja, com 0s e 1s.

Os conjuntos pré-definidos de instruções, que são traduzidos pelo compilador, passaram então a ser chamados de **linguagens de programação**.



Alan Turing, 1912-1954

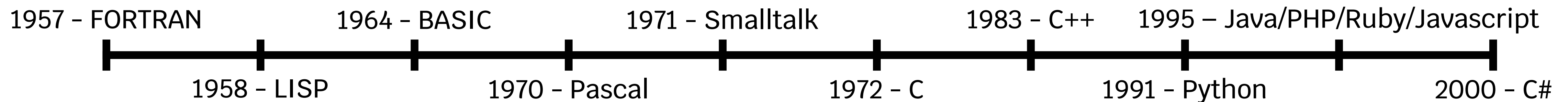
- Considerado o pai da computação moderna e da inteligência artificial
- Responsável por quebrar a segurança da máquina Enigma, usada pelo exército nazista na II Guerra Mundial
- Propôs a máquina de Turing, computador teórico que é considerado até hoje como o melhor modelo de um computador de uso geral
- Propôs tabelas de instruções que automaticamente converteriam a escrita decimal em códigos numéricos, permitindo que computadores fizessem muito mais que operações matemáticas

Linguagens de programação

As linguagens que utilizavam os conceitos propostos por Hopper (com os compiladores) e Turing (com as tabelas de instruções) passaram a ser chamadas de **linguagens de alto nível**, ou seja, linguagens que são facilmente lidas e interpretadas por seres humanos.

As **linguagens de baixo nível** são aquelas legíveis apenas para máquinas. Atualmente, temos pouquíssimos programadores de linguagens de baixo nível.

Com o avanço da capacidade dos hardwares, começaram a surgir algumas linguagens de alto nível **interpretadas**, ao invés de compiladas.



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))
```

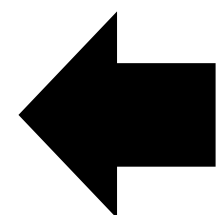
```
1  for cont in range(10):  
2  |  print(str(cont))  
3  |  print(str(cont))
```


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.

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**.



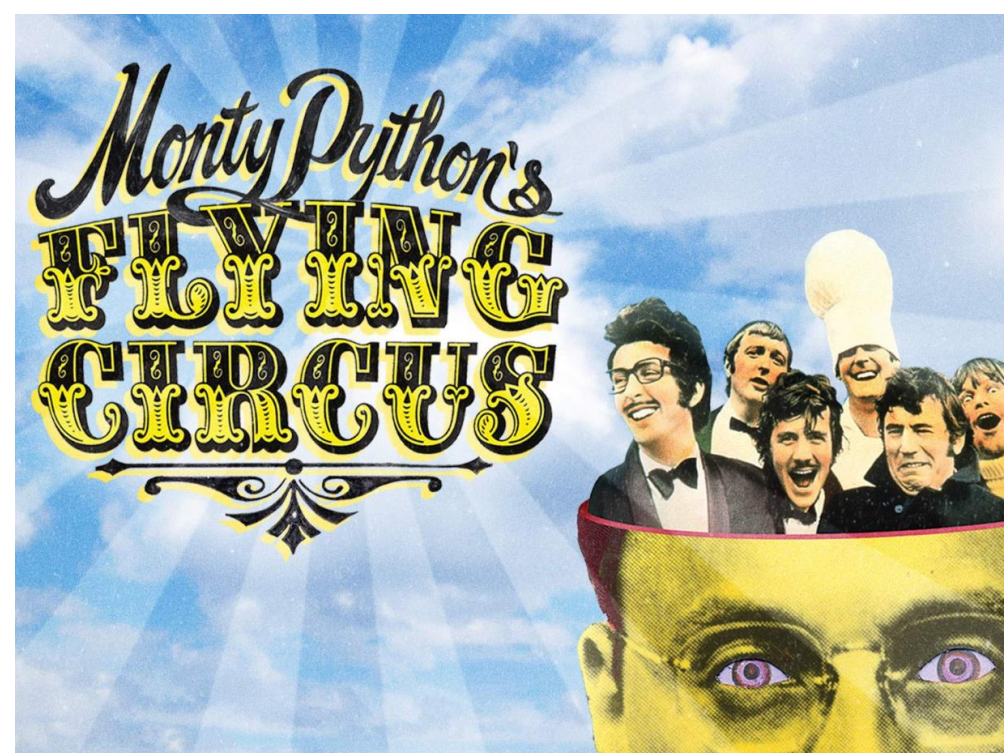
Sobre Python

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



Sobre Python

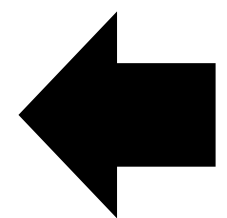
Características do Python:

- 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

Sobre Python

Principais áreas de atuação:

- Implementação de algoritmos de Inteligência Artificial, em particular no processamento de imagens;
- Desenvolvimento back-end para aplicações web, através de bibliotecas como Django;
- Criação de projetos de Ciência de Dados, com manipulação e análise de grandes volumes de dados;
- Desenvolvimento de soluções para Internet das Coisas, como sensores e dispositivos para automação residencial;
- Uso no dia-a-dia de equipes de desenvolvimento em geral, através da criação de scripts de automação de *build* e integração contínua.



Configurando o ambiente Python

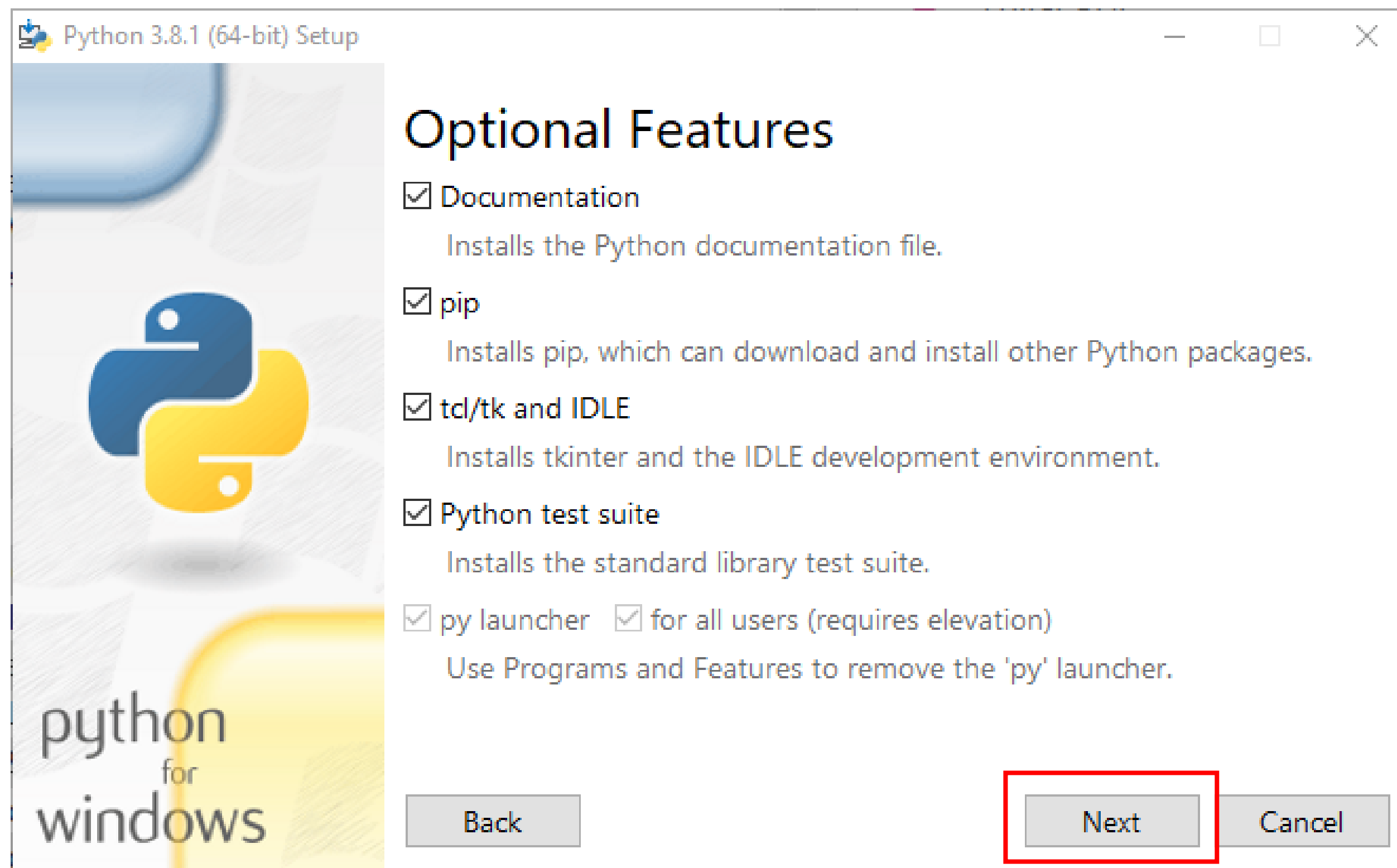
Como instalar Python?

- 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”.



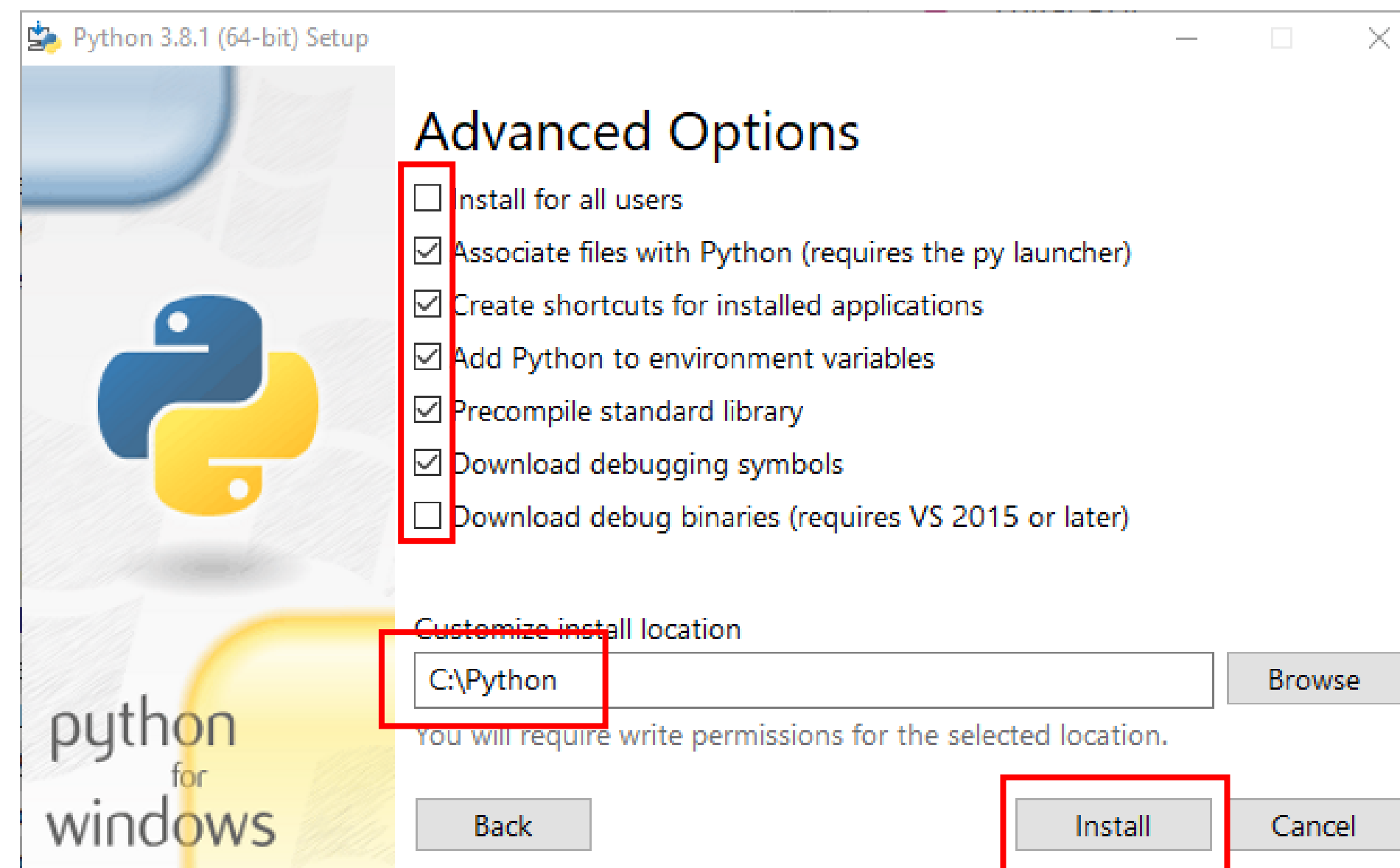
Como instalar Python?

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




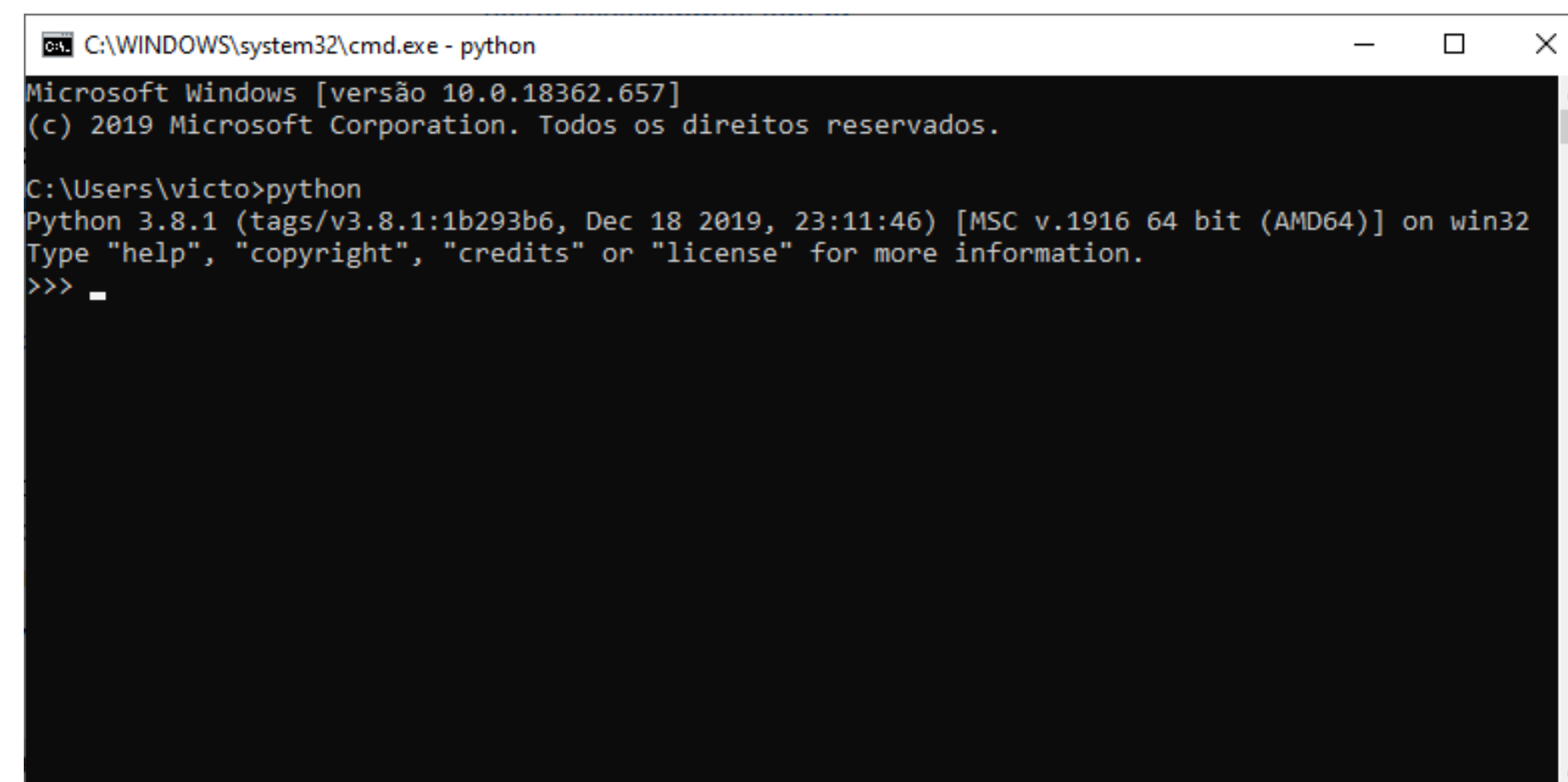
Como instalar Python?

- 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.



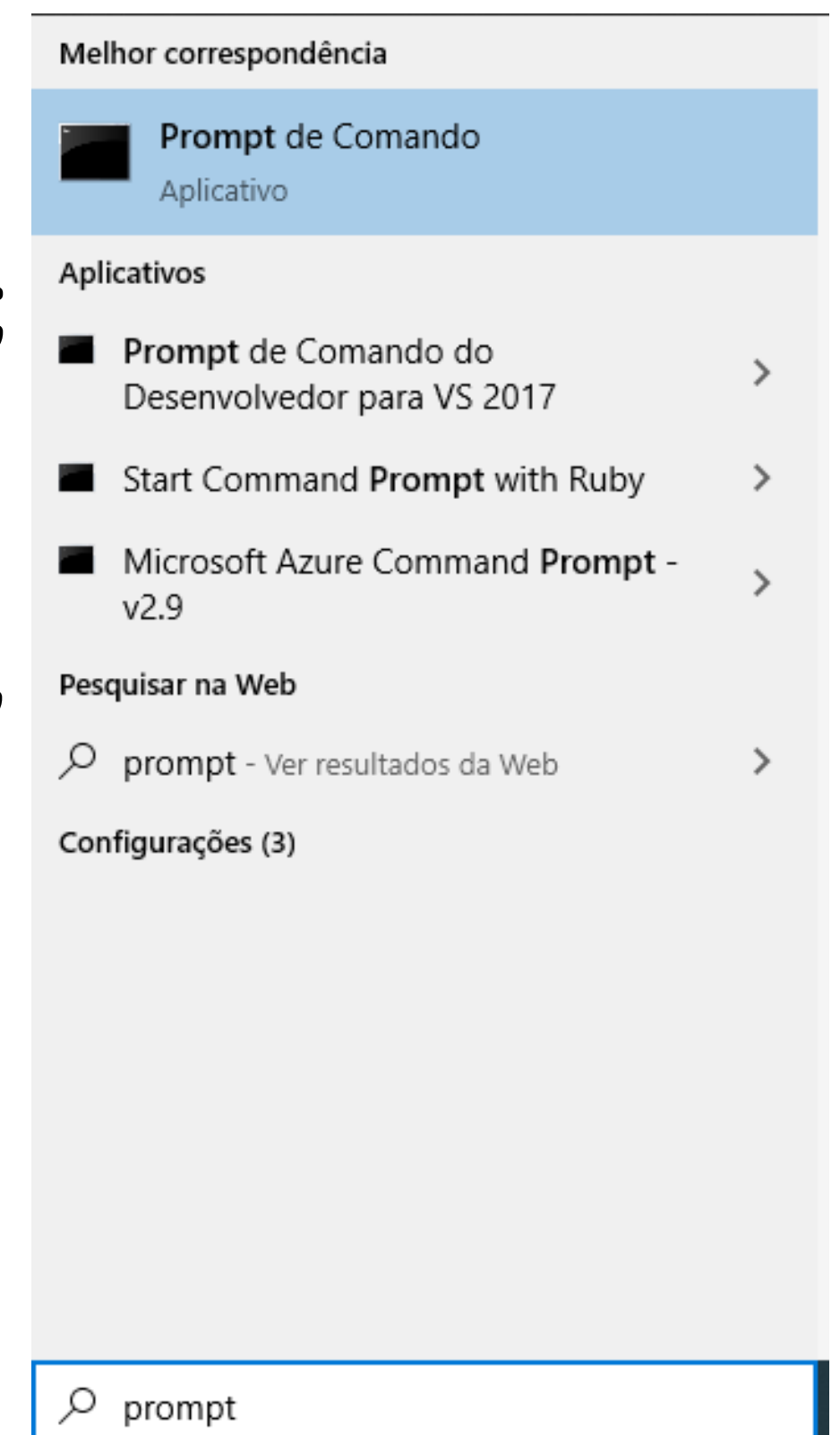
Como instalar Python?

- 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.



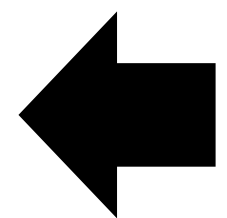
```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [versão 10.0.18362.657]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

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

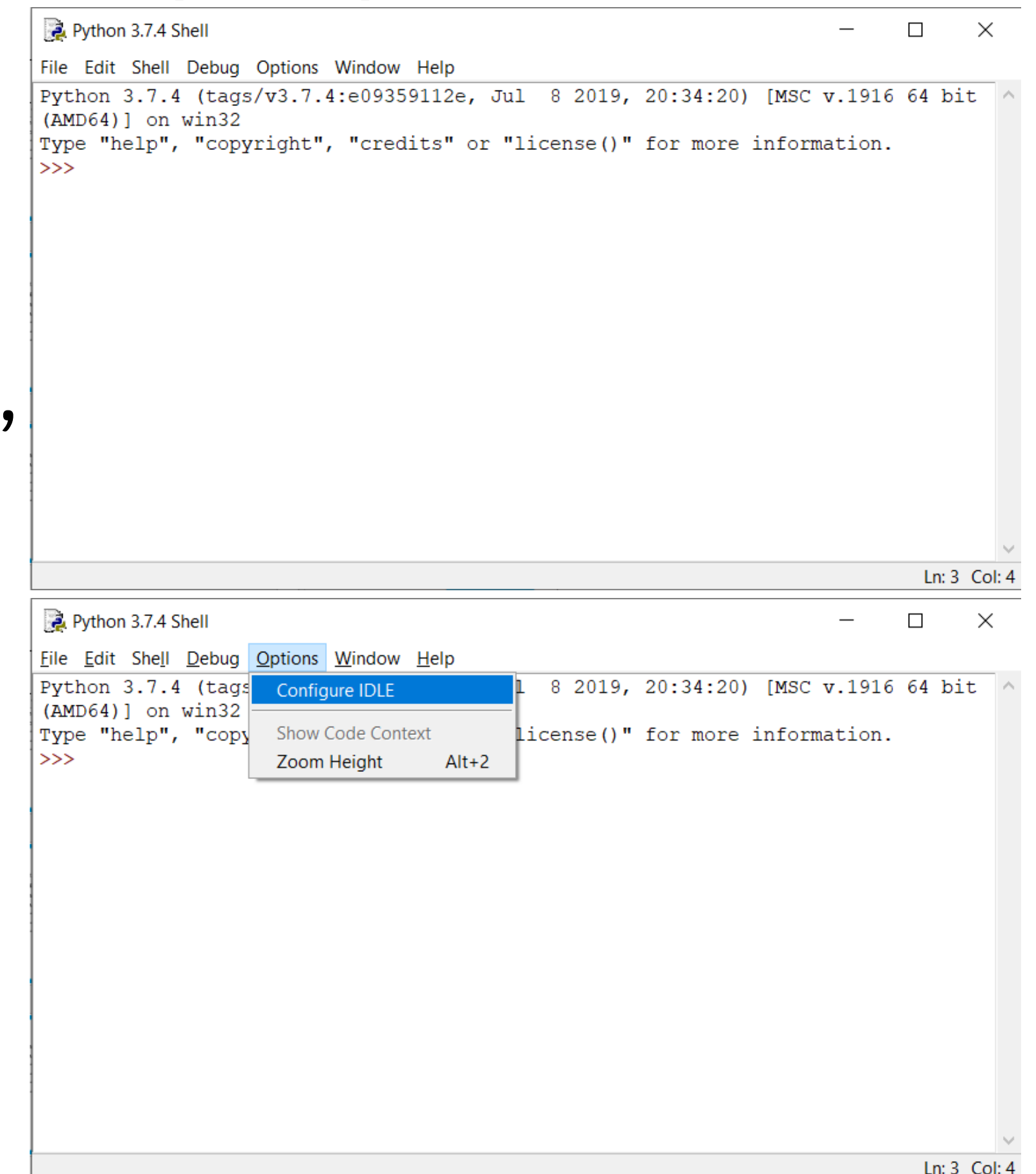
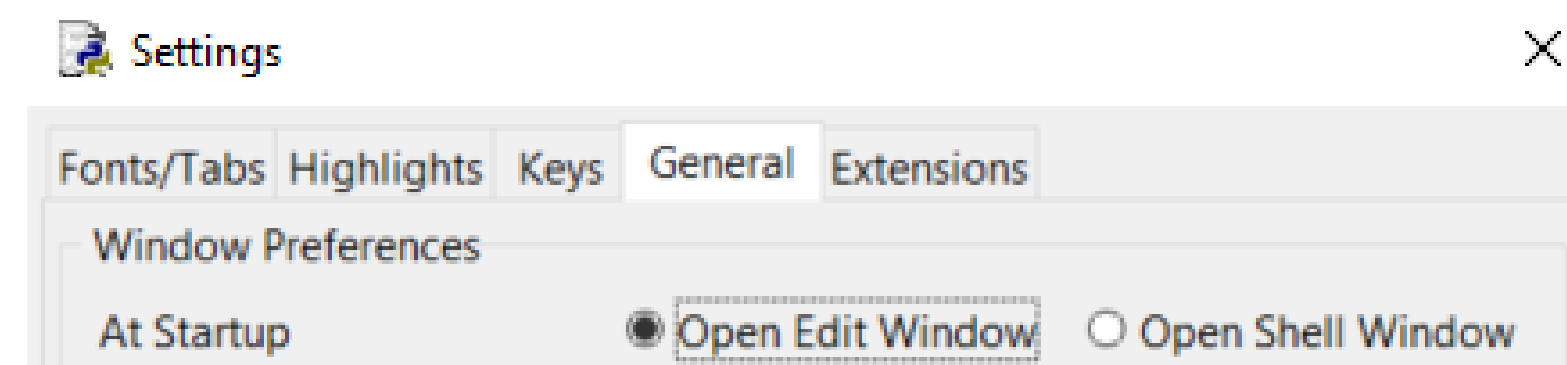
1. Antes de abrir o programa, abra o Windows Explorer, clique em **Este Computador** e, em seguida, no drive do seu computador (C: ou D:, dependendo da sua máquina);
2. Neste diretório, crie uma pasta chamada **Projetos**. Esta pasta será usada para armazenar todos os seus projetos de software;
3. Dentro 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

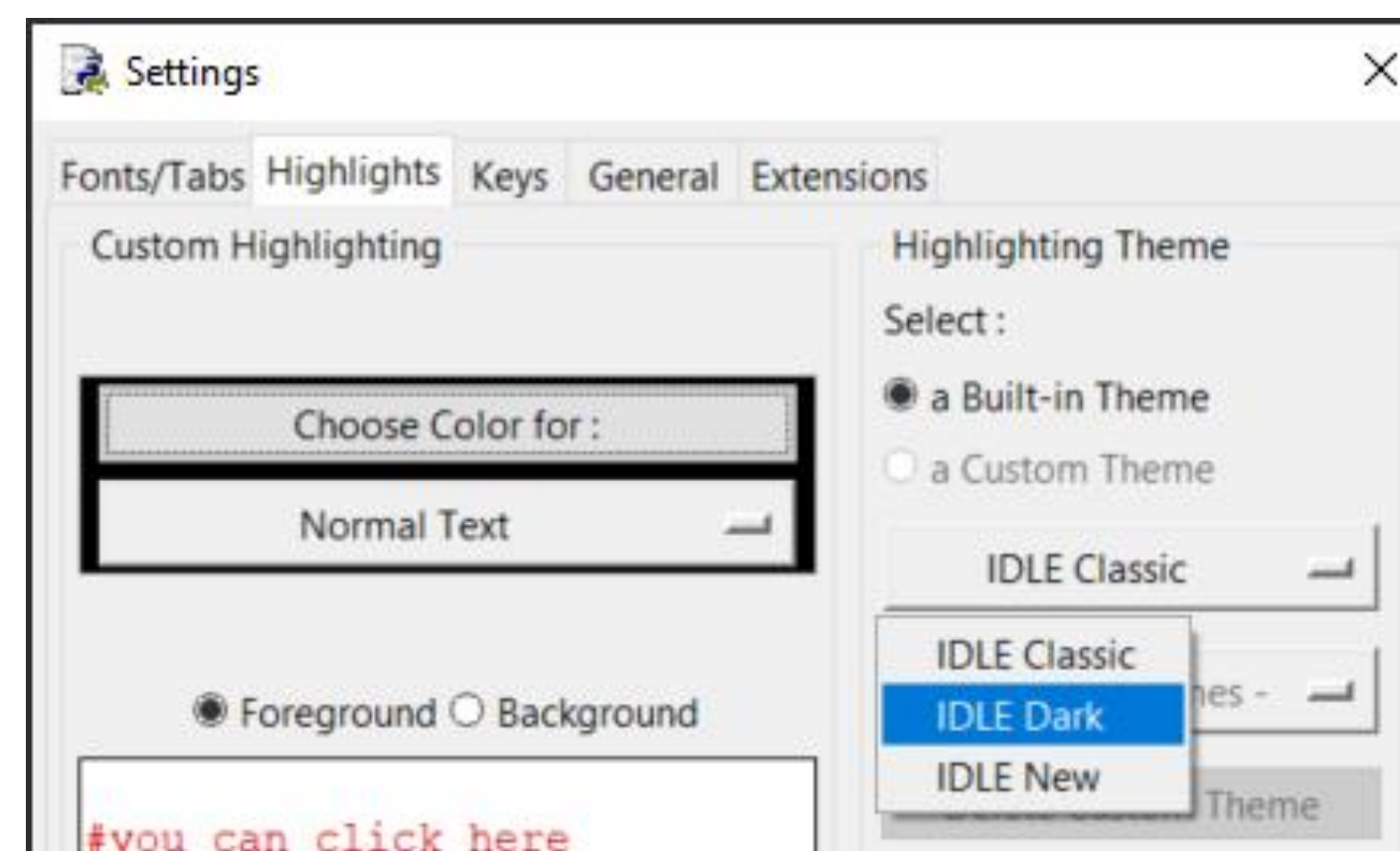
Configurando o IDLE

- 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



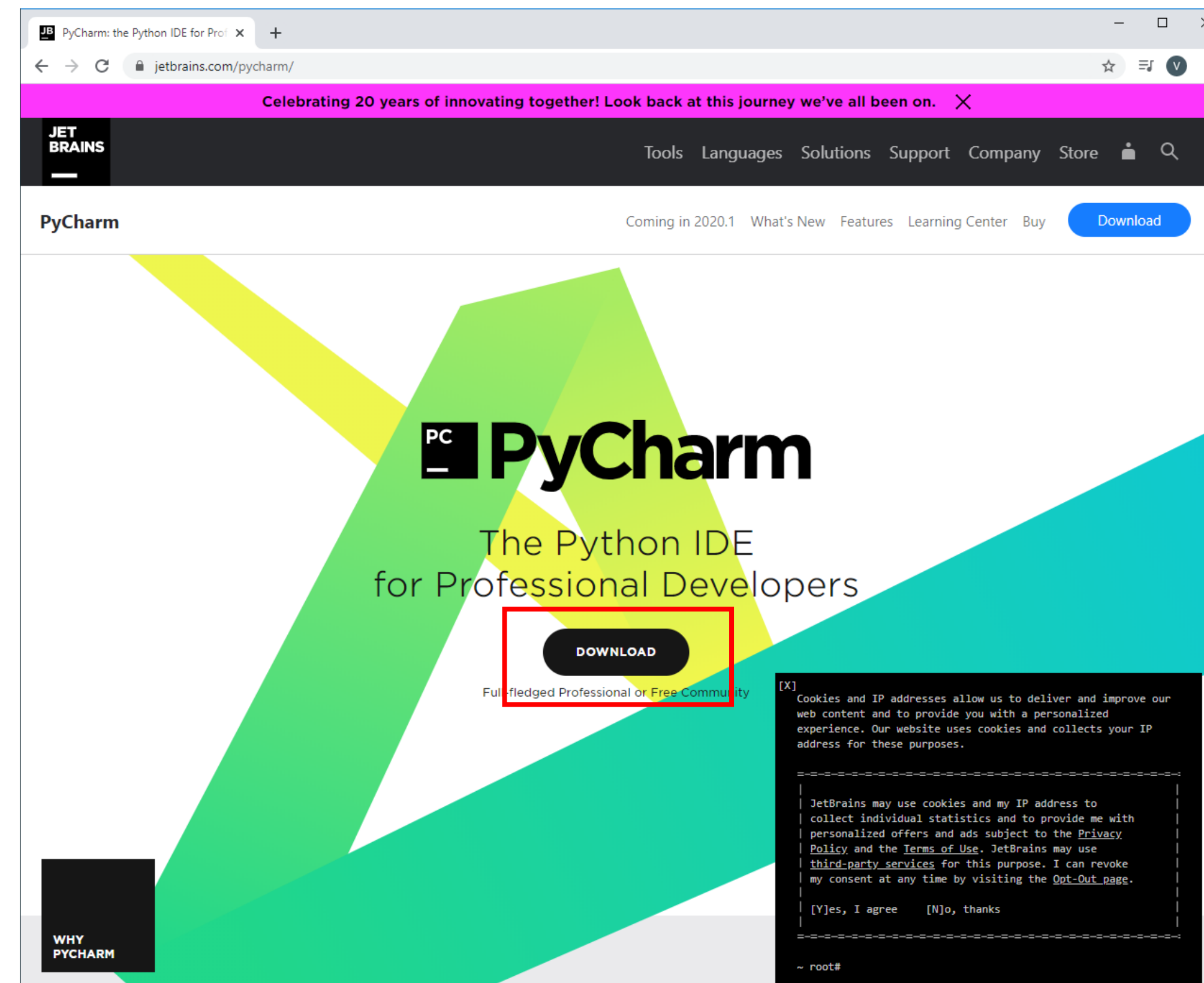
Configurando o IDLE

- 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



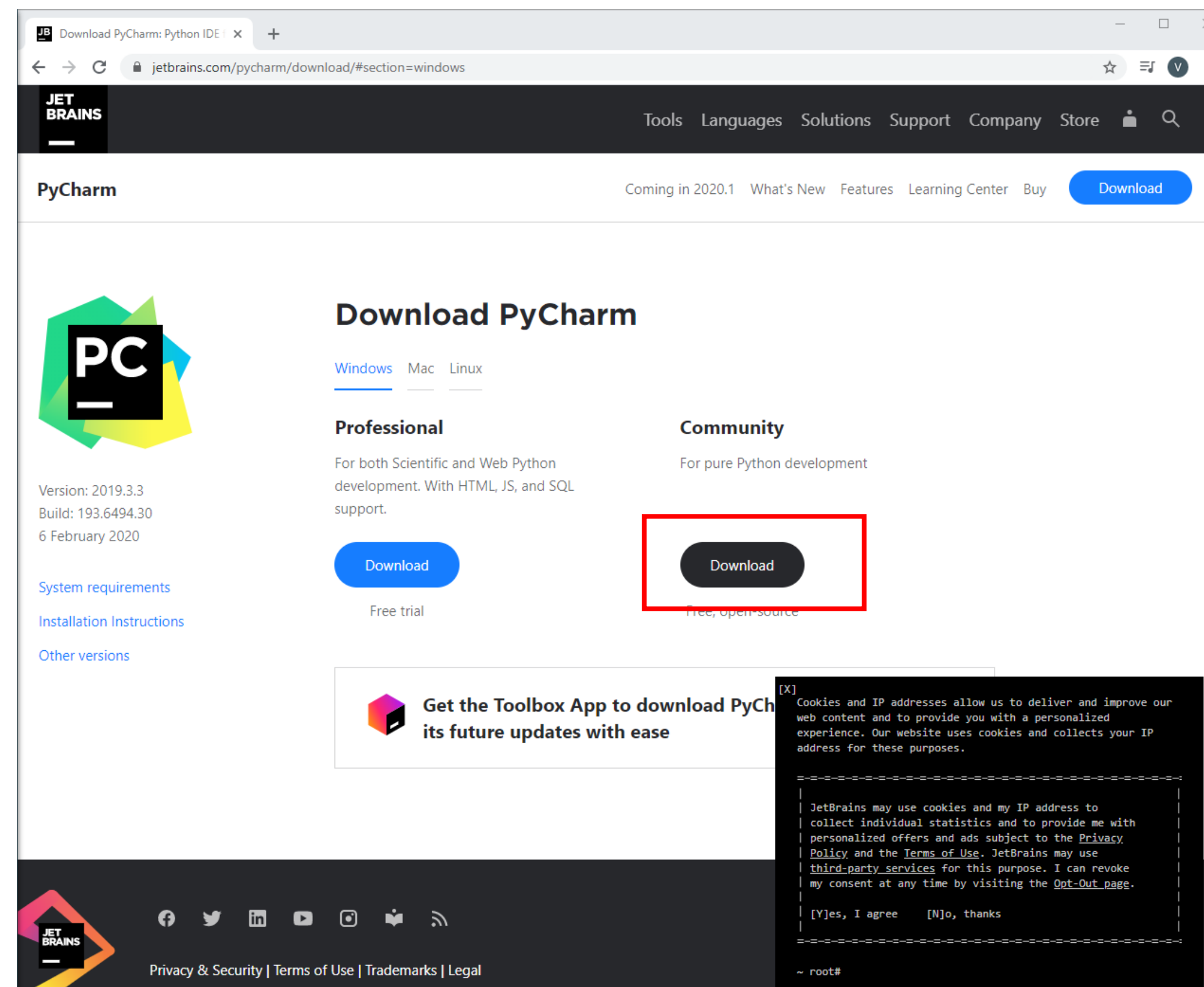
Instalando e configurando o PyCharm

- 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”.



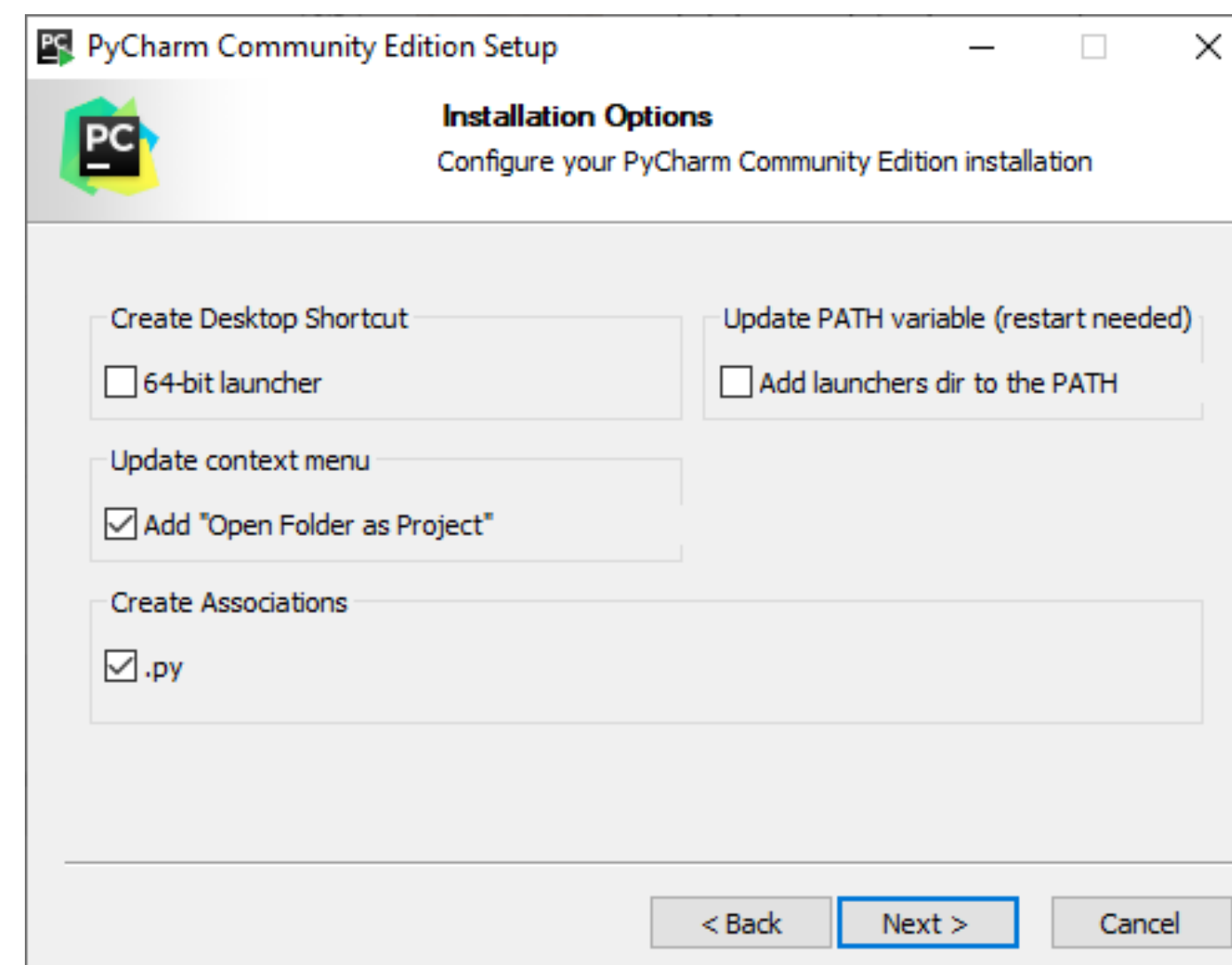
Instalando e configurando o PyCharm

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



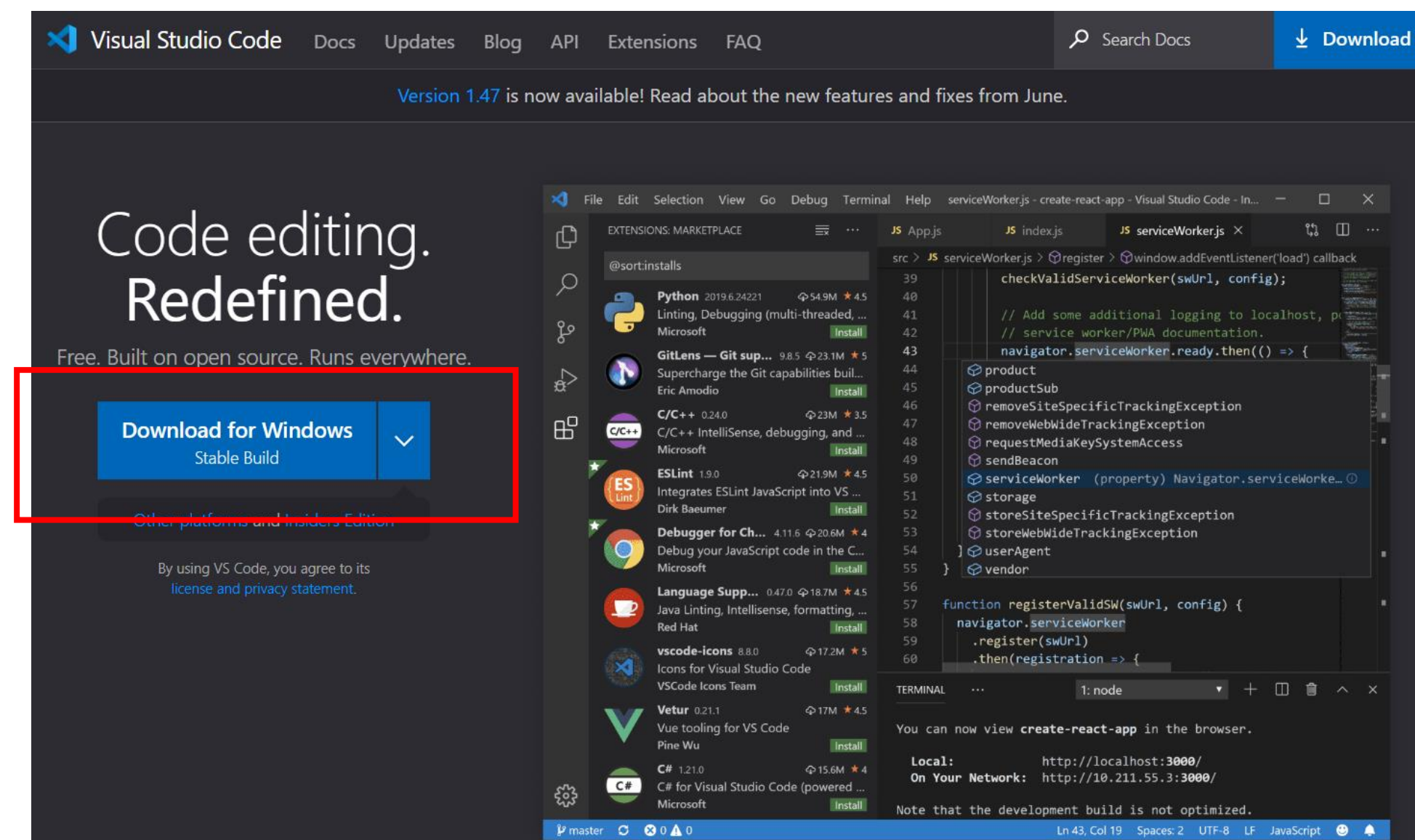
Instalando e configurando o PyCharm

- 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.



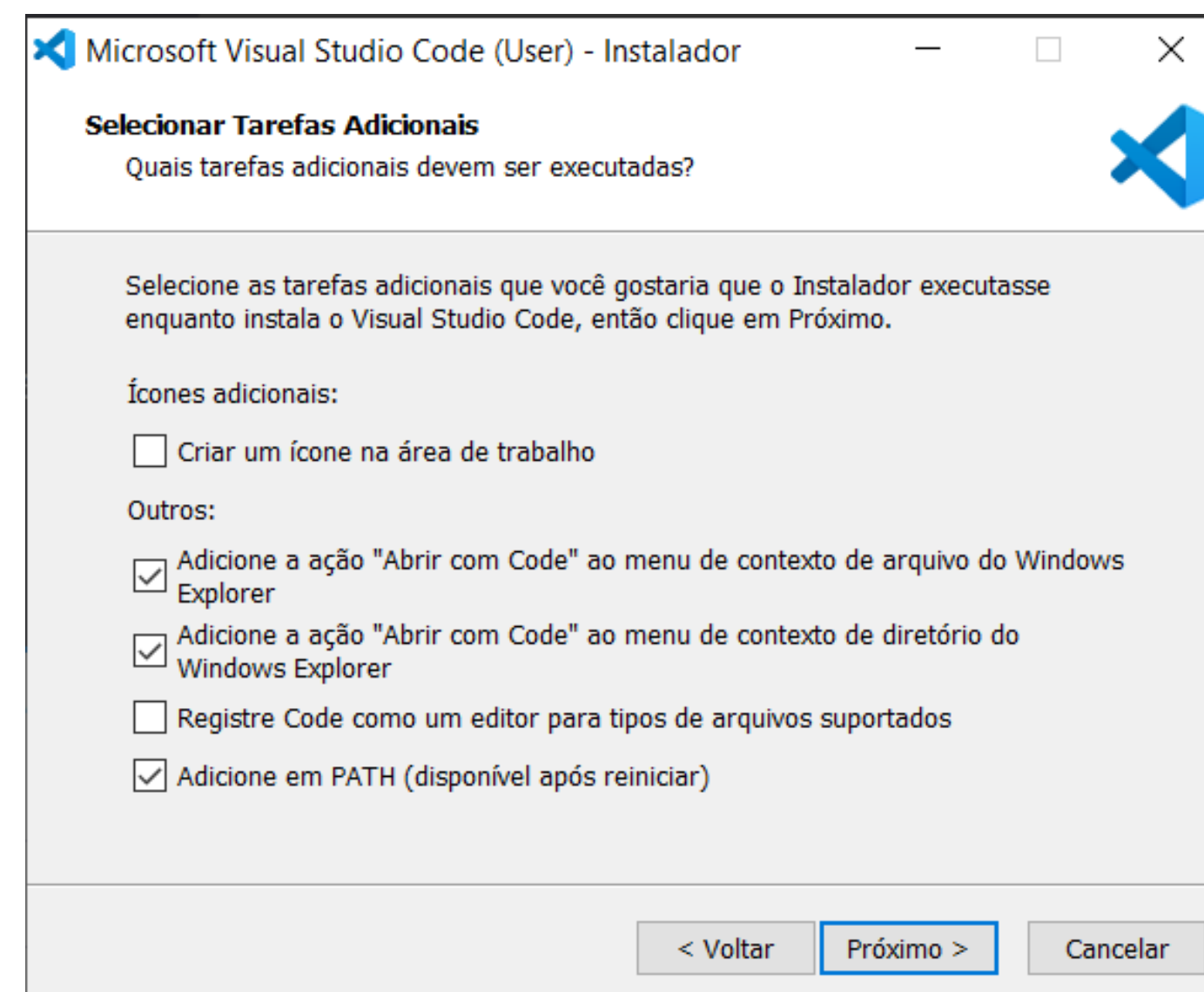
Instalando e configurando o VSCode

- 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

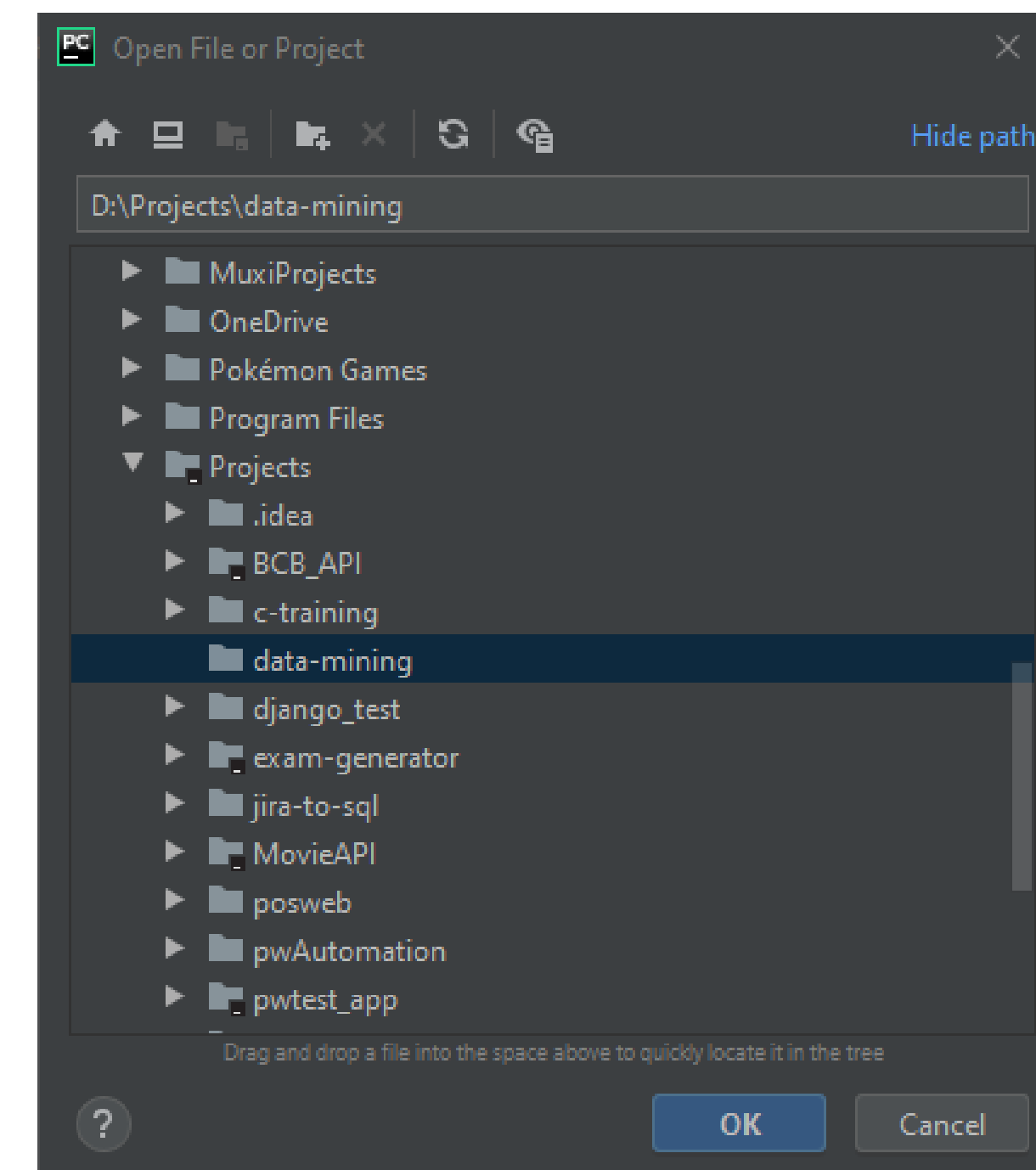
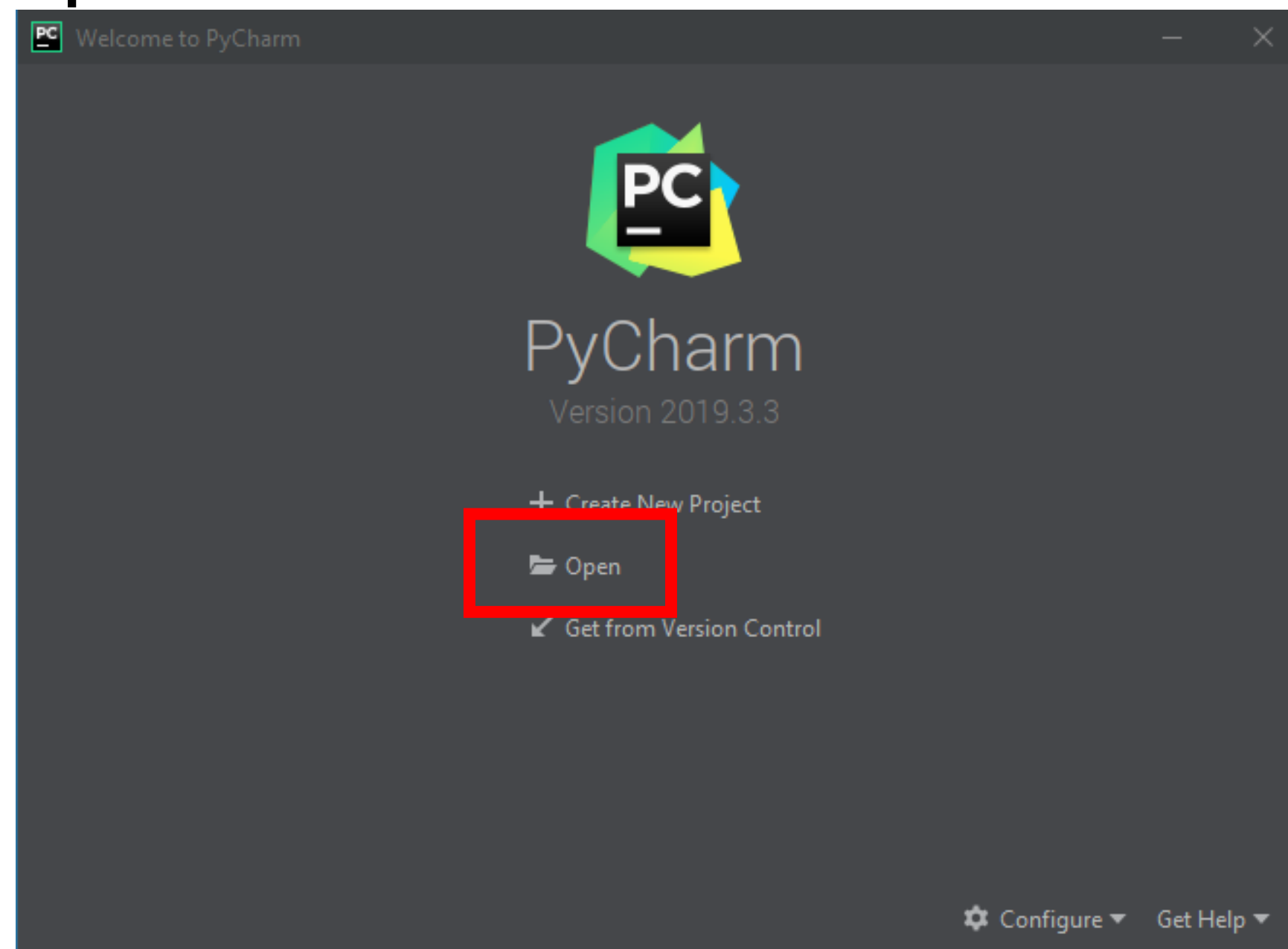
- Abra o instalador, e após aceitar o acordo de licença e clicar em “Próximo”, clique em “Próximo” novamente até chegar na tela “Selecionar Tarefas Adicionais”
- Marque as opções abaixo e clique em “Próximo” e depois em “Instalar”



Utilizando o PyCharm

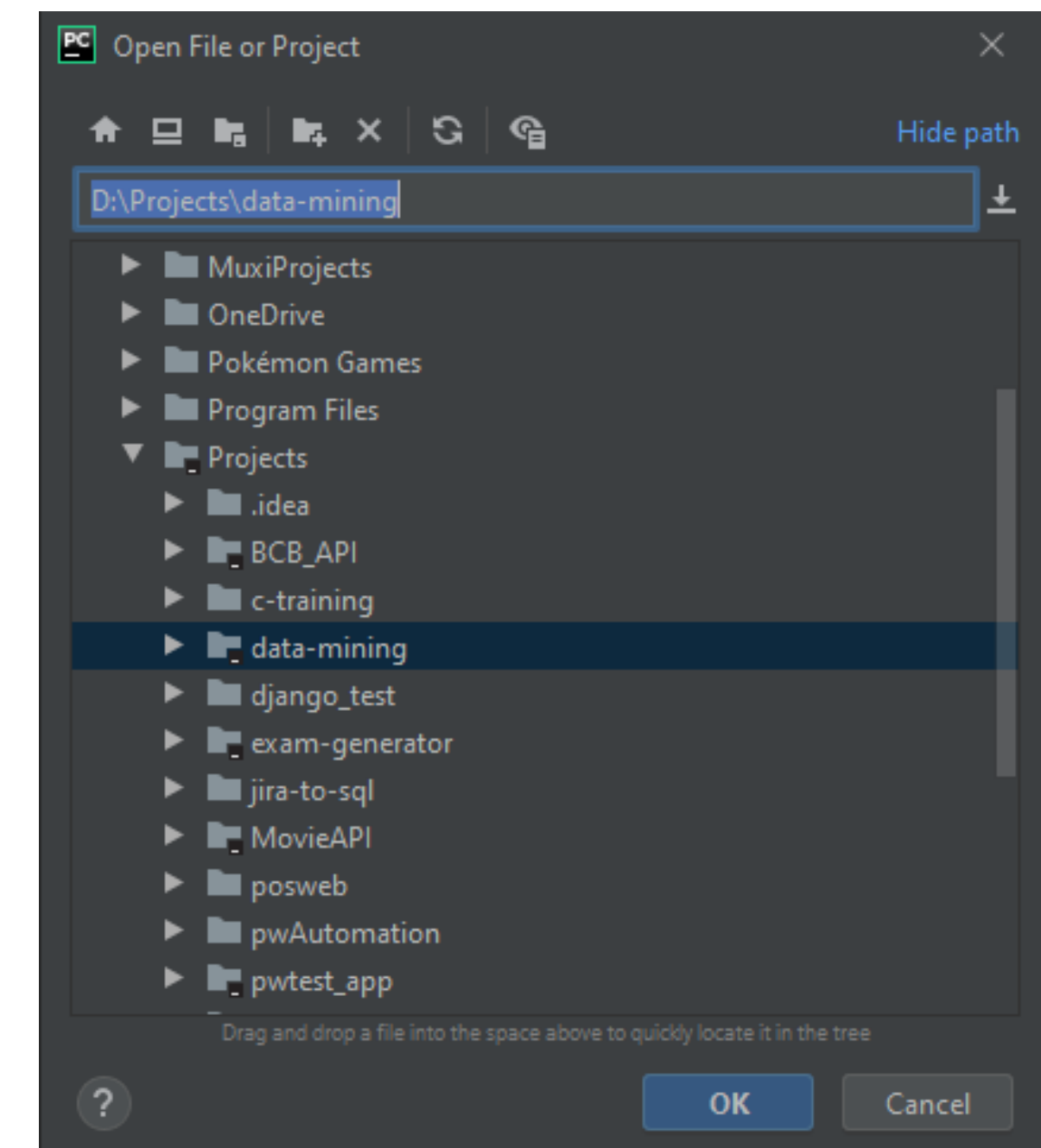
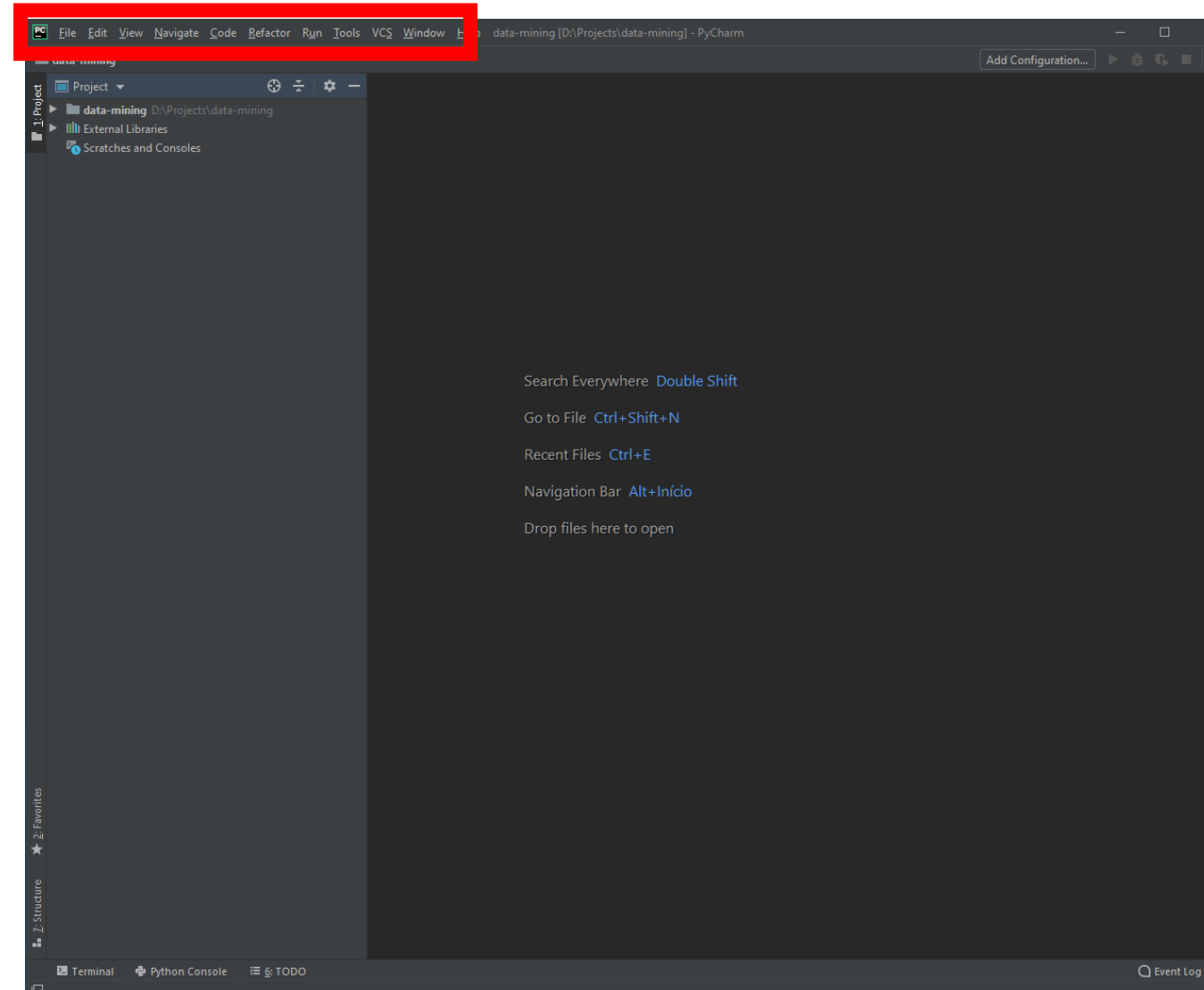
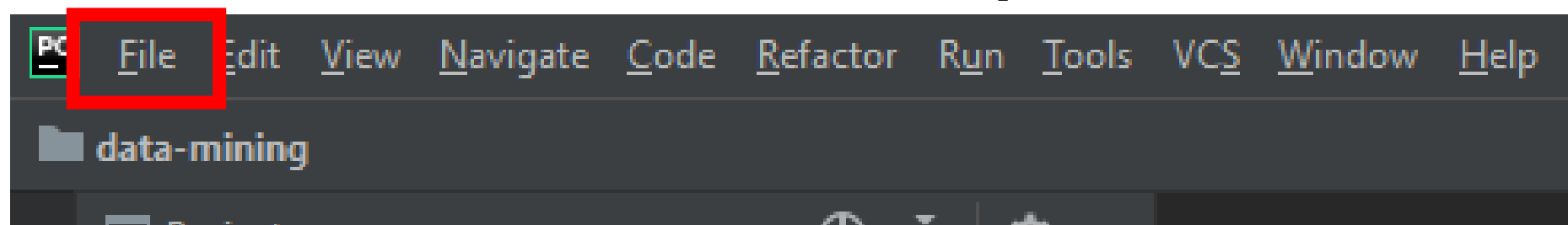
Iniciando o PyCharm

- 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**;

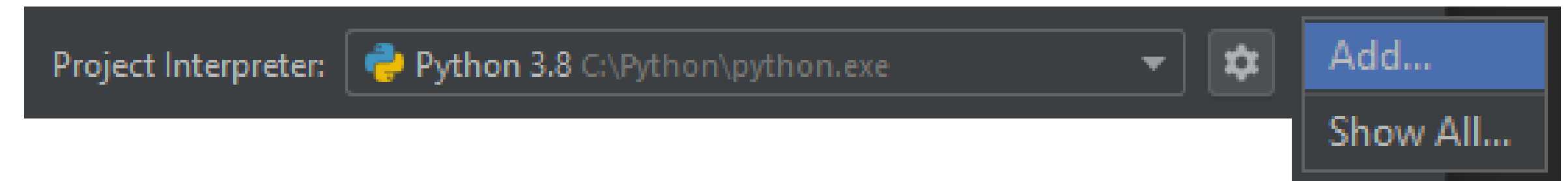


Algumas configurações do PyCharm

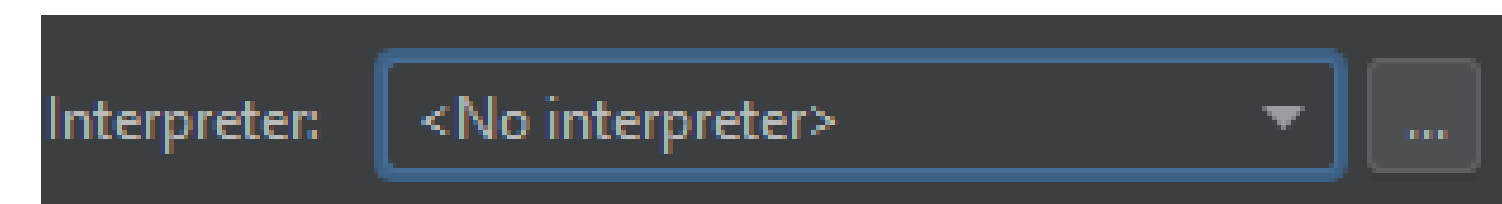
- Mudar o tema para escuro:
 - Clique em **File > Settings**;
 - Na janela que aparecer, procure por **Appearance & Behavior > Appearance**. No campo **Theme**, selecione o tema desejado (minha sugestão é o **Darcula**);
 - Clique em **Ok** para fechar a tela de configurações.

Algumas configurações do PyCharm

- Configurar o interpretador de Python:



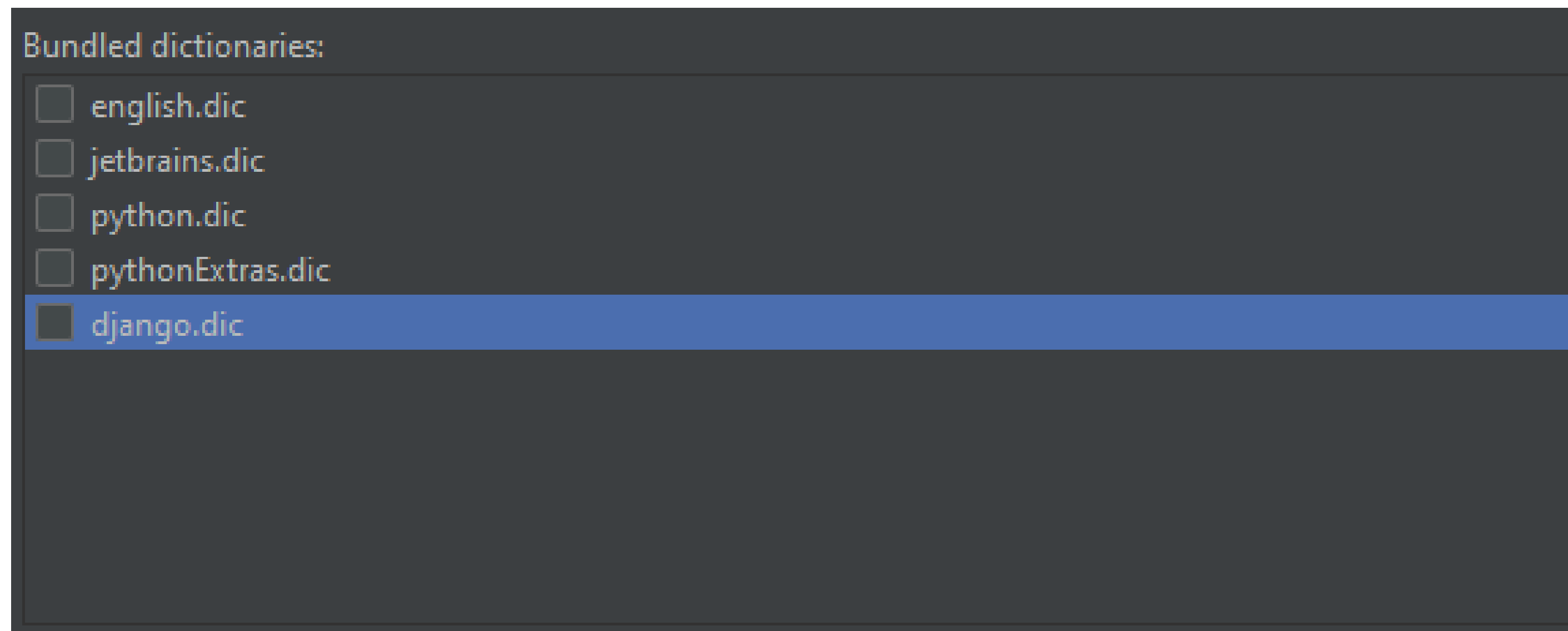
- Clique em **File > Settings**;
- Na janela que aparecer, procure por **Project: <nome-do-projeto> > Project Interpreter**. No campo **Project Interpreter**, verifique se o Python informado é o instalado (no meu caso, **C:\Python\python.exe**). Caso não seja, clique na engrenagem e em **Add...**;
- 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**.

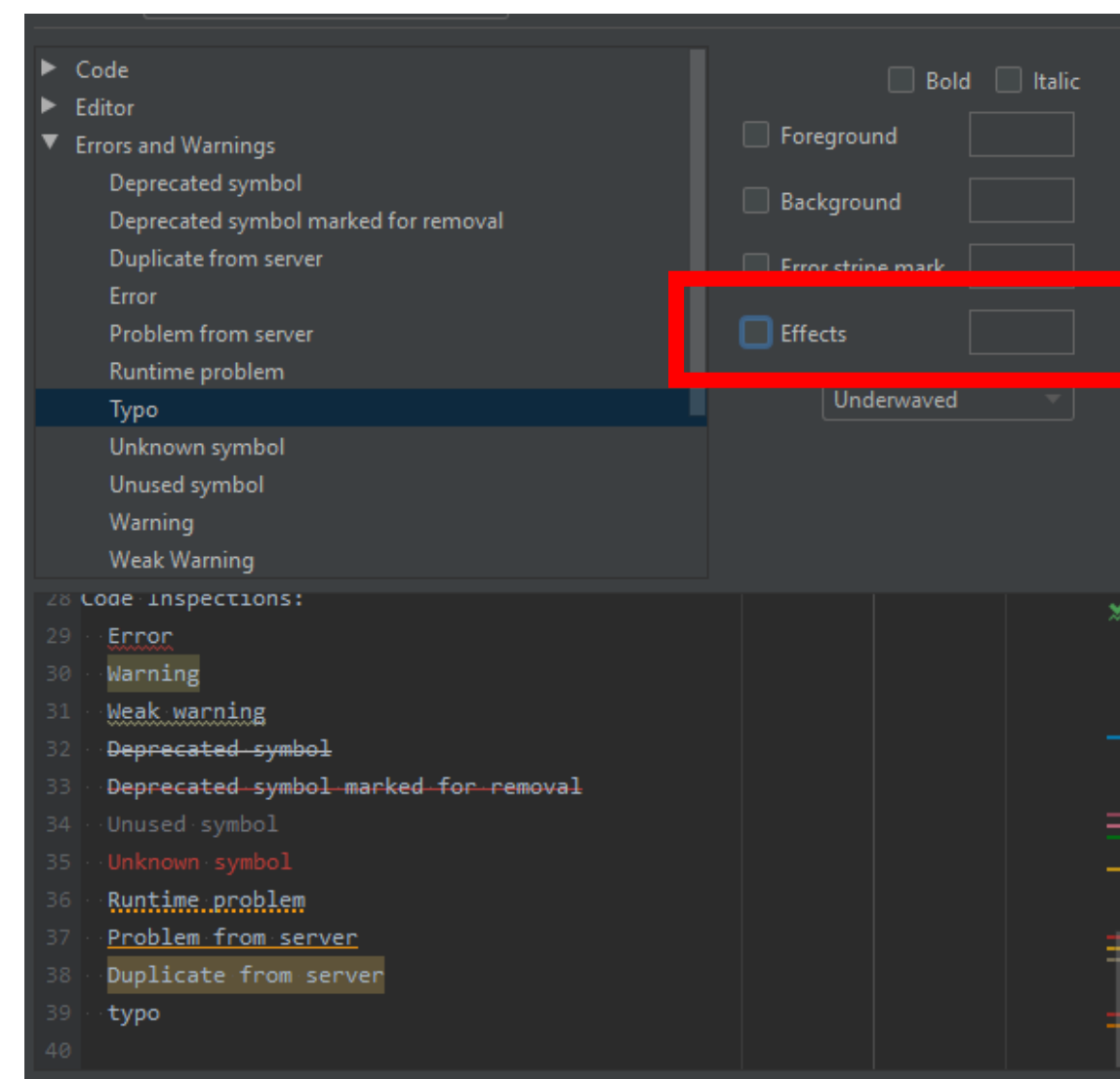
Algumas configurações do PyCharm

- 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.



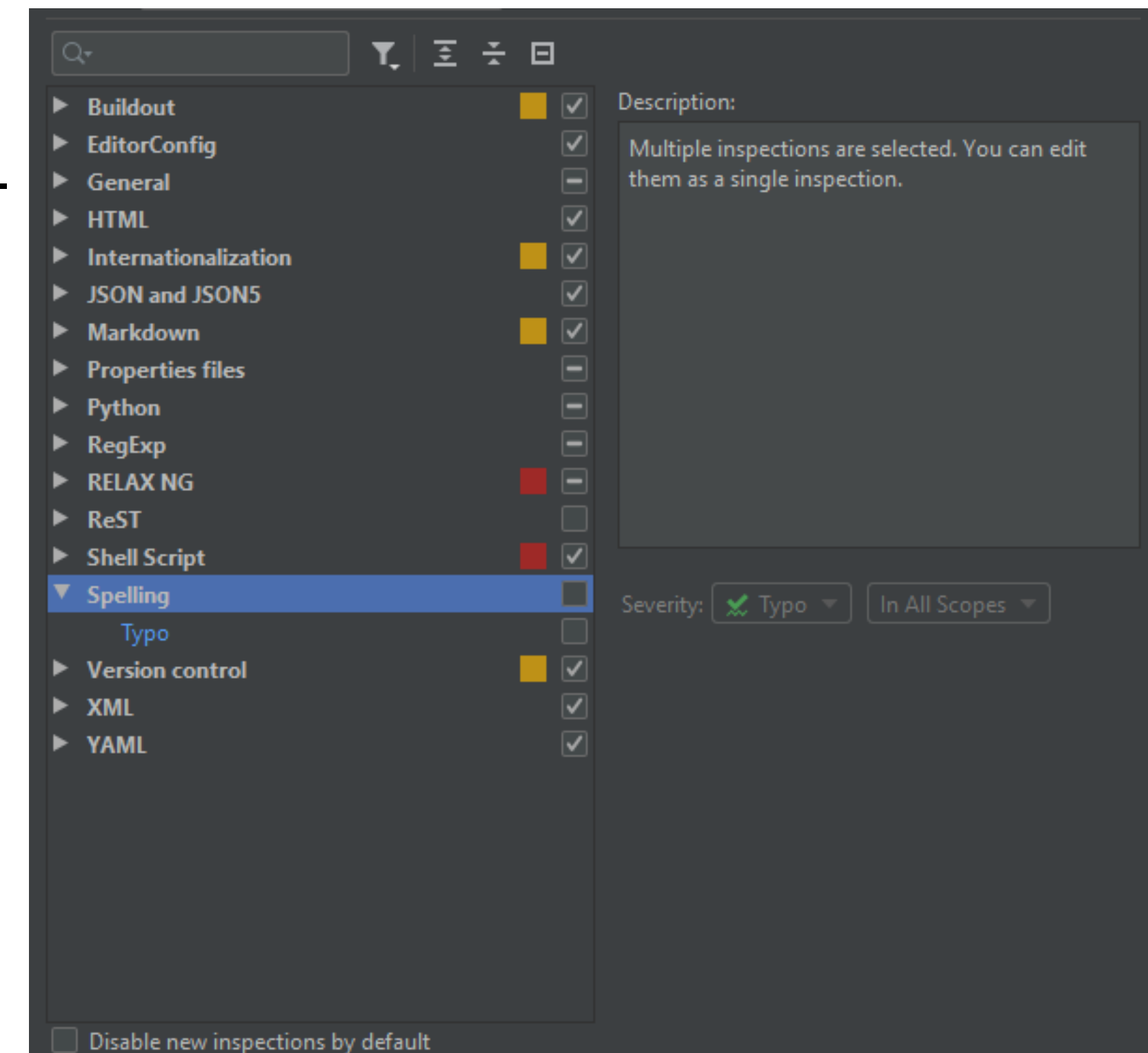
Algumas configurações do PyCharm

- 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.



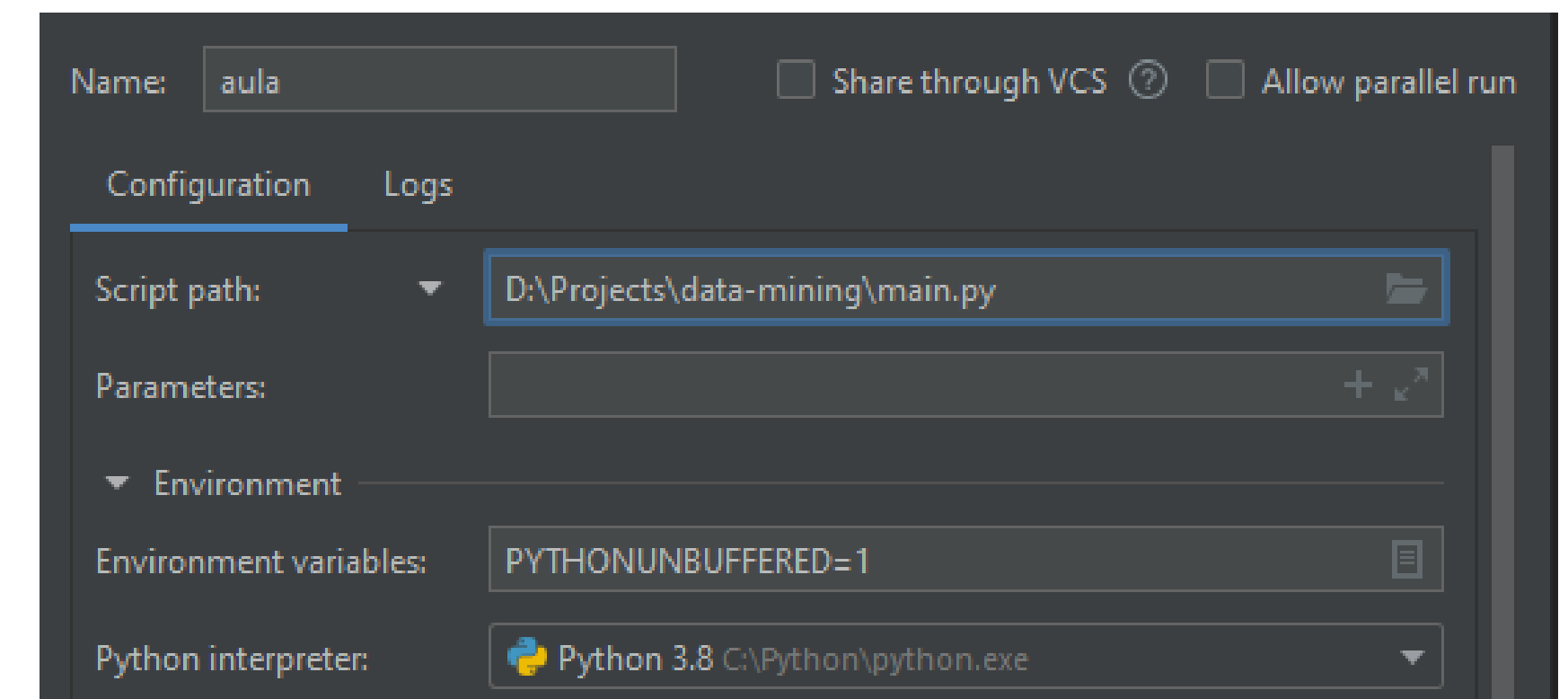
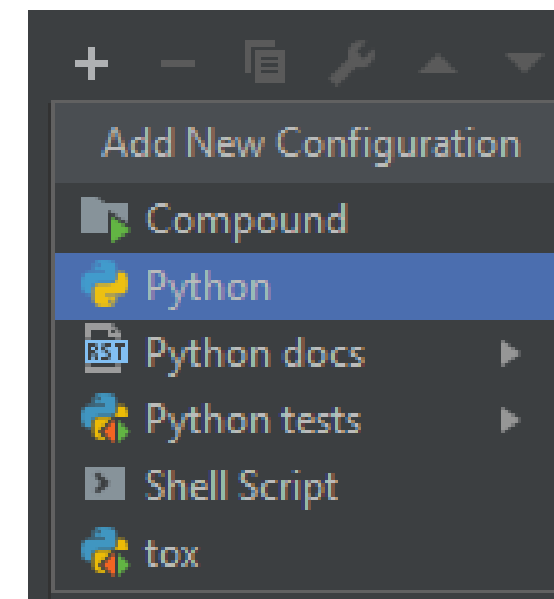
Algumas configurações do PyCharm

- Ajustar os destaques de inspeções:
 - Clique em **File > Settings**;
 - Na janela que aparecer, procure por **Editor > Inspections**. Desmarque o campo **Spelling**;
 - Na seção **Python**, recomendo *desmarcar* algumas opções para simplificar o aprendizado:
 - *Boolean variable check can be simplified*;
 - *Chained comparisons can be simplified*;
 - *Comparison with None performed with equality operators*.
 - Clique em **Ok** para sair das configurações.



Algumas configurações do PyCharm

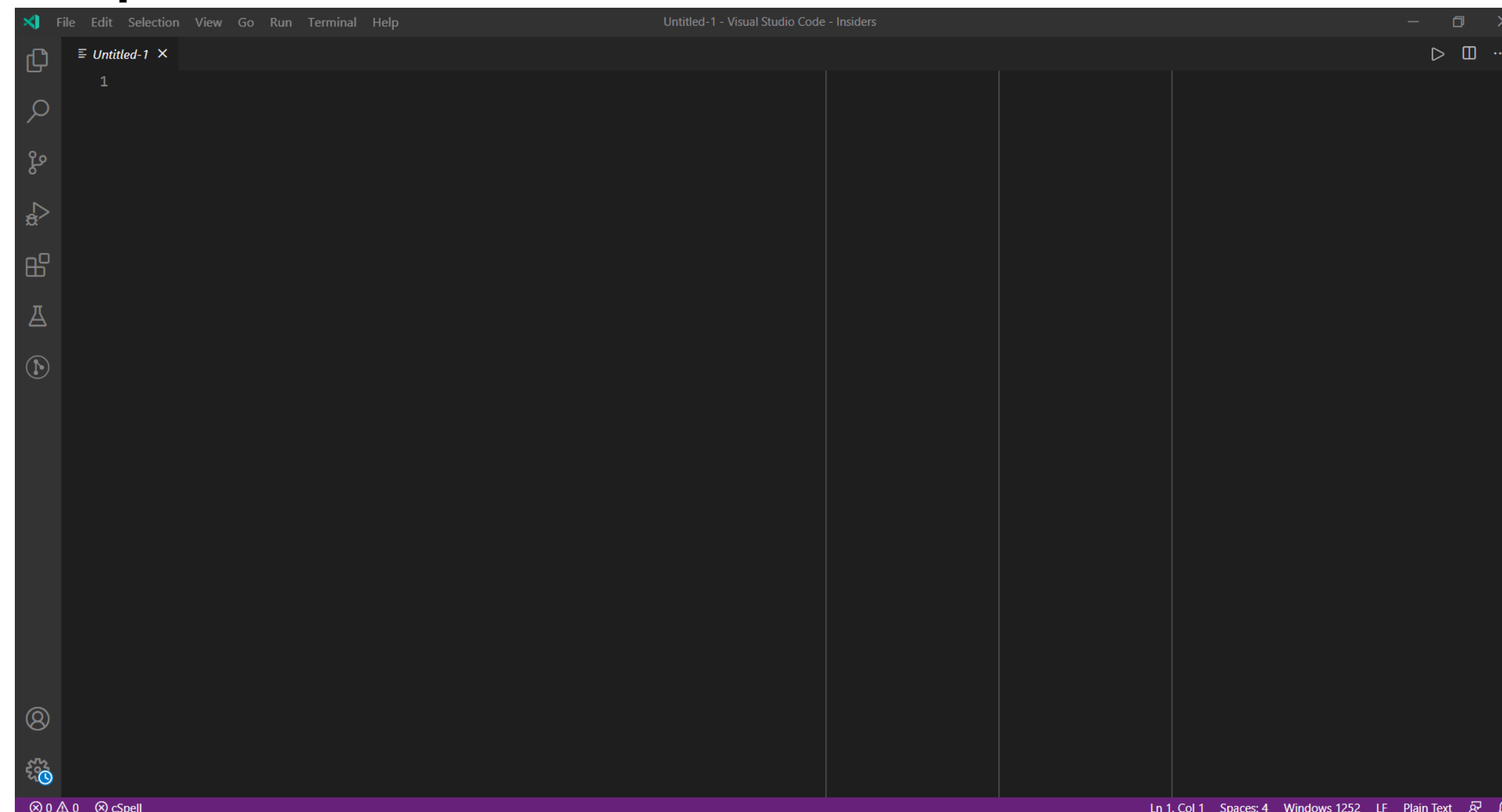
- Configurar uma determinada execução de código:
 - Garanta que as configurações do slide Configurar o interpretador de Python foram executadas;
 - Aperte **Alt + Shift + F10** > **Edit Configurations**, ou na barra de tarefas clique em **Run** > **Edit Configurations**;
 - Na janela que aparecer, no 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**.



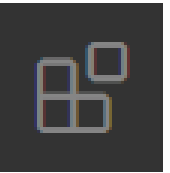
Utilizando o VSCode

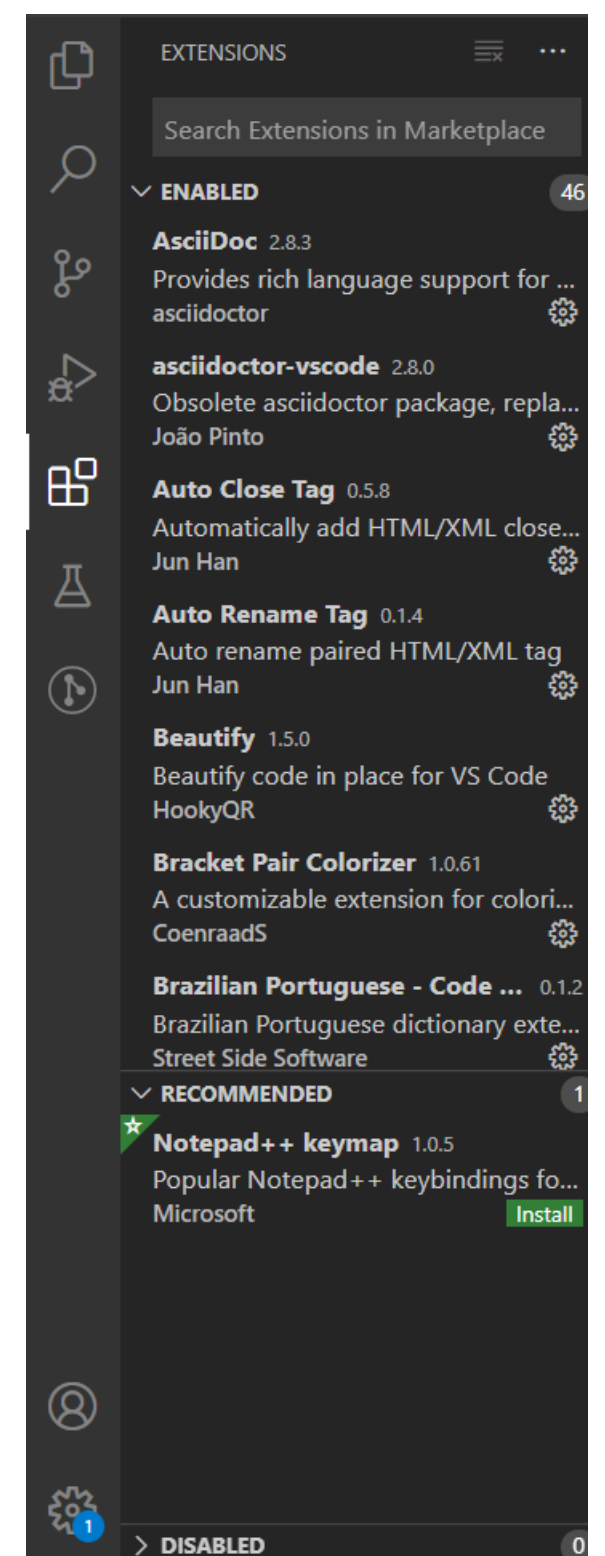
Iniciando o VSCode

- 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

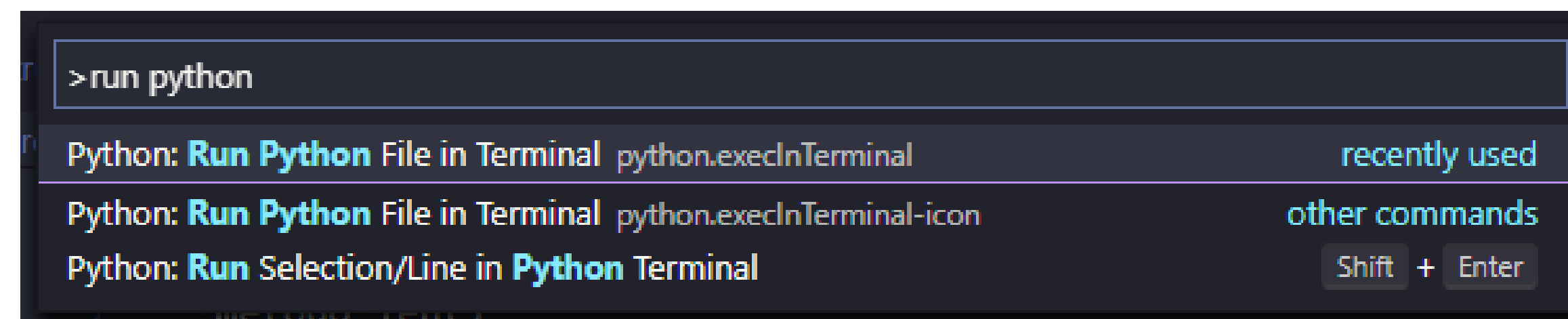
- **Atalhos interessantes no VSCode:**
 - F12: Vai para a declaração de uma variável ou função
 - F12 F12: Mostra todos os usos de uma variável ou função
 - Alt + → ou Alt + ← : Retrocede ou avança na última posição do cursor
 - Ctrl + S: Salva um arquivo
 - Ctrl + P: Abre um arquivo diretamente
 - Ctrl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

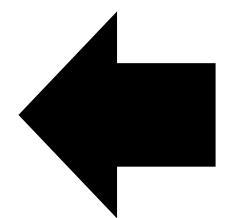
Iniciando o VSCode

- **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.





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
```

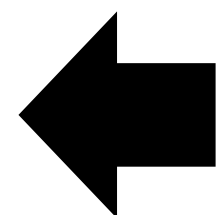
- Lembre-se de substituir o caminho do Python para o caminho instalado no seu computador, e altere **pylint** para o pacote escolhido.

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>



Configurando o pylint

Configurando o pylint

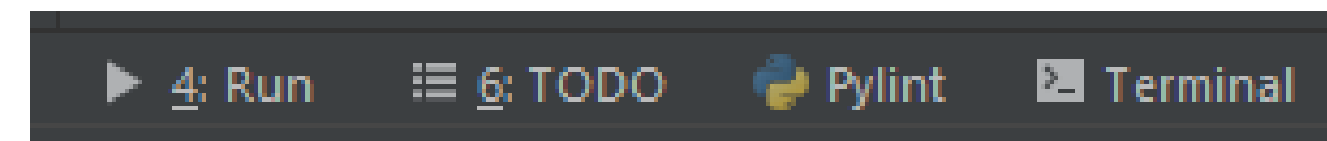
- O pylint é um pacote do Python que auxilia na adequação do código aos padrões de programação da comunidade Python
- Para instalar o pylint...
 - ...para MacOS veja [aqui](#)
 - ...para Windows veja [aqui](#)
- Para executar o pylint basta entrar com o comando **pylint <caminho>**, com o caminho do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

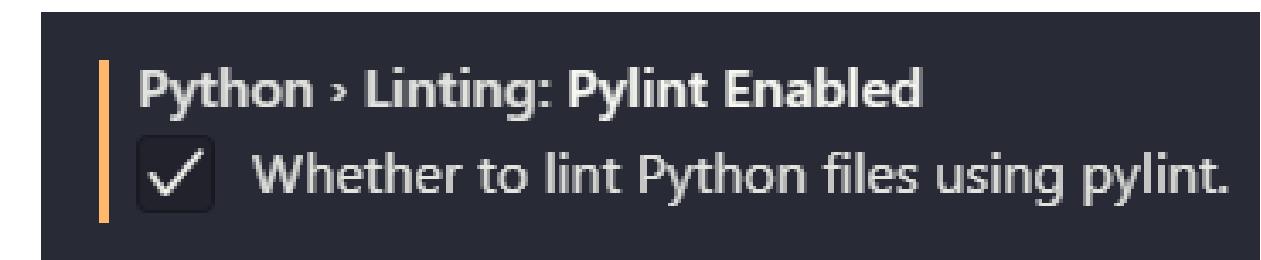
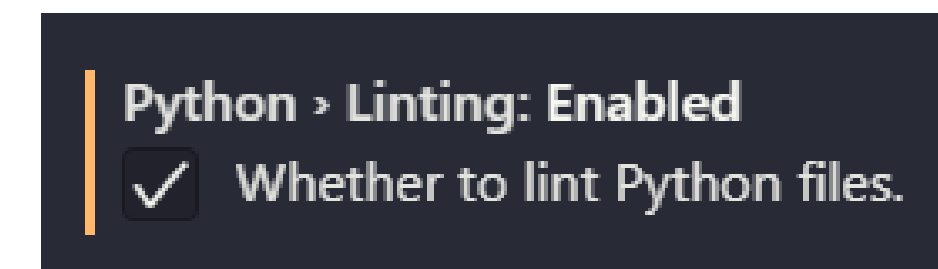
Configurando o pylint

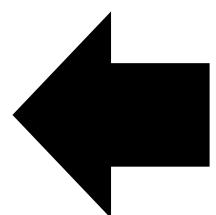
- O pylint ainda pode ser rodado direto no terminal do VSCode ou do PyCharm, basta abrir o terminal e seguir os mesmos passos mencionados anteriormente. No entanto, os ambos os IDEs fornecem meios de utilizar o pylint diretamente durante a implementação do código.
- **Usando o pylint no PyCharm:**
 - 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”.



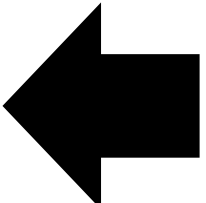
Configurando o pylint

- 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.





Algoritmos e Lógica de Programação



O que são algoritmos?

Uma resposta geral seria "um conjunto de etapas para executar uma tarefa". Todos nós executamos algoritmos na nossa vida diária, desde escovar os dentes ao dirigir um carro.

Para fritar um ovo, por exemplo, temos uma sequência de etapas:

- Pegar um ovo na geladeira;
- Selecionar uma frigideira e colocá-la no fogão;
- Acender a chama do fogão;
- Colocar um pouco de óleo na frigideira;
- Quebrar o ovo e temperar;
- Aguardar N segundos e servir.

O que são algoritmos?

Basicamente, para qualquer tarefa do dia-a-dia, precisamos de uma sequência de etapas, que normalmente já está internalizado nas nossas cabeças.

Da mesma forma como precisamos de algoritmos para termos um funcionamento básico no nosso dia, os computadores também o precisam.

- Rotas de aplicativos como o Waze;
- Compra de produtos na Amazon;
- Uso de um micro-ondas.

A principal diferença entre os algoritmos dos humanos e os de computador é que, nos humanos, conseguimos tolerar uma pouca precisão nas descrições.

- Dizer "quebrar um ovo" pode ser extremamente ambíguo!

O que são algoritmos?

Portanto, quando definimos um algoritmo de computador, precisamos descrever como sendo **"um conjunto de etapas para executar uma tarefa descrita com precisão suficiente para que um computador possa executá-la"**.

Usualmente, um bom algoritmo de computador precisa de algumas características:

- É preciso: todas as etapas e pré-condições precisam ser bem definidas;
- Deve possuir um nível adequado de corretude;
- Precisa ser eficiente.

Como representar algoritmos

Terminologia usada:

- Procedimentos
- Chamada
- Parâmetros
- Retorno

```
PROCEDIMENTO incremento(valor)
-----
# Entrada: parâmetro valor é um número.
# Saída: o parâmetro valor acrescido de uma unidade.
-----
1. Encerra o procedimento, retornando valor + 1.
```

```
PROCEDIMENTO algoritmo_inutil()
-----
# Entrada: Nada.
# Saída: Nada.
-----
1. Atribui 1 à variável numero.
2. Atribui à variável numero o retorno do procedimento incremento(numero).
3. Encerra o procedimento.
```

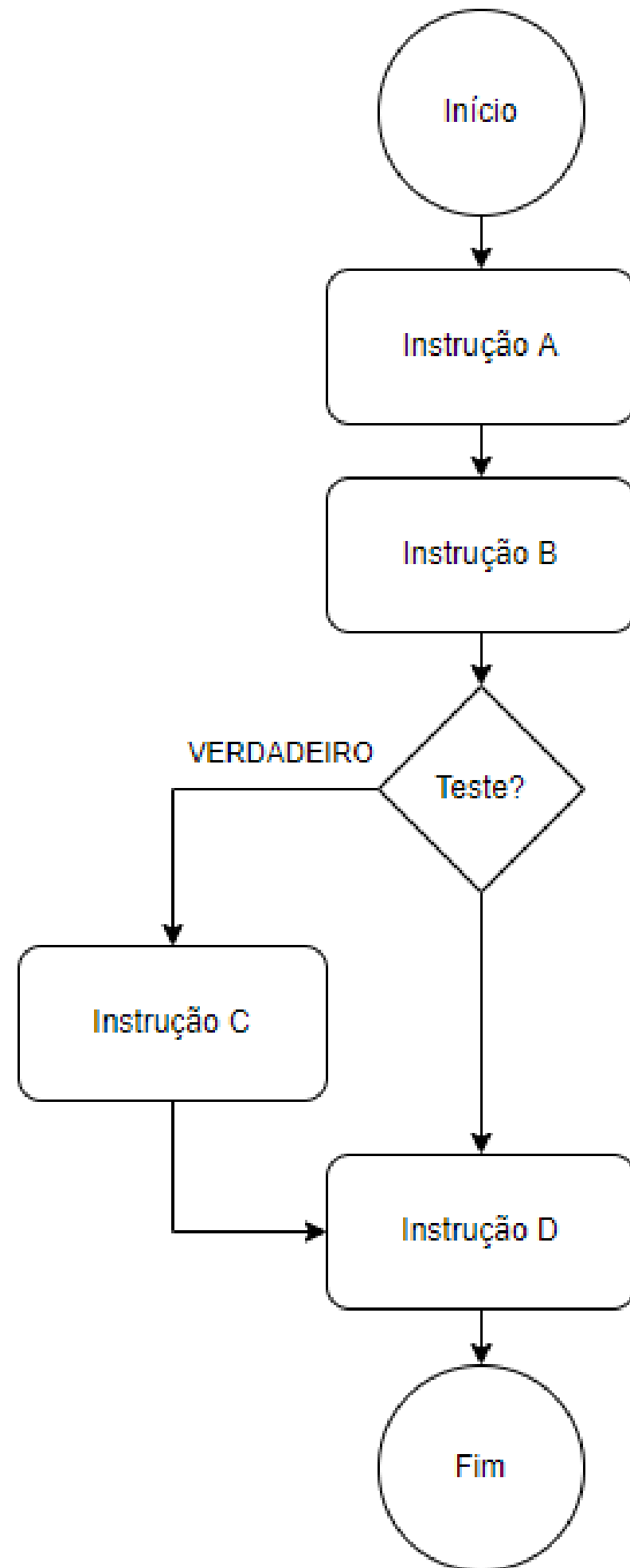
Como representar algoritmos

Noção de arranjos:

- Arranjo agrega dados do mesmo tipo em uma única entidade;
- Considere o arranjo ao lado, que chamamos de `lista_compras`;
- O elemento de índice 3 na tabela é Carne, e representamos por `lista_compras[3]`;
- Arranjos começam pelo índice 0!
- O tempo para localizar qualquer elemento de um arranjo é o mesmo, independentemente de sua posição.

| Índice | Item para comprar |
|--------|-------------------|
| 0 | Pão |
| 1 | Queijo |
| 2 | Ovo |
| 3 | Carne |
| 4 | Manteiga |

Estruturas de seleção



```
PROCEDIMENTO exibe_nota(nota)
```

```
-----  
# Entrada: parâmetro nota é um número entre 0 e 10.
```

```
# Saída: nenhuma.  
-----
```

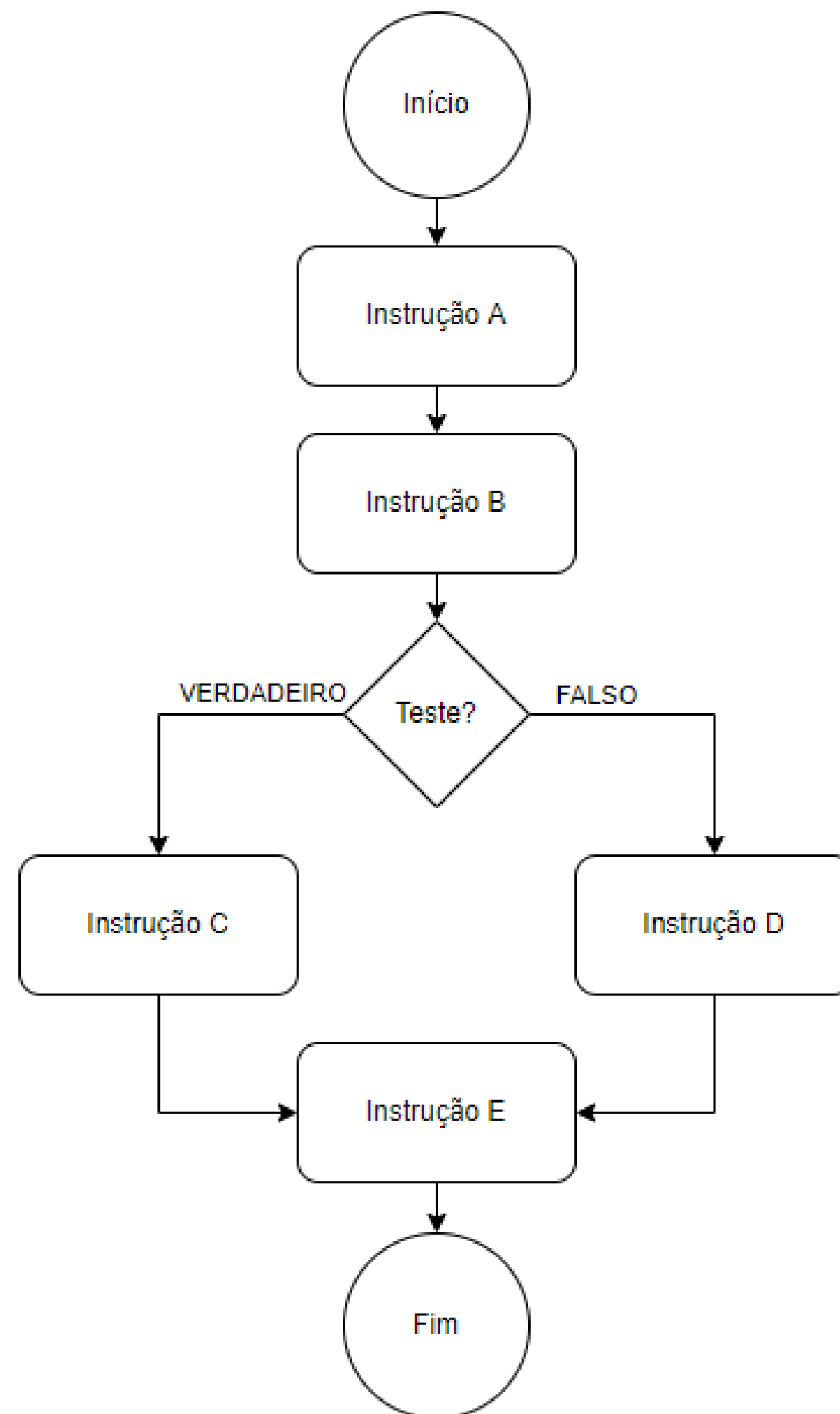
```
1. Se nota > 9.0 faça:
```

```
    1.1. Exiba na tela a mensagem "Parabéns!".
```

```
2. Exiba na tela a mensagem "Sua nota foi ", seguida pelo valor de nota.
```

```
3. Encerre o procedimento.
```


Estruturas de seleção



```
PROCEDIMENTO exhibe_nota(nota)
```

```
-----  
# Entrada: parâmetro nota é um número entre 0 e 10.  
# Saída: nenhuma.  
-----
```

```
1. Se nota > 9.0 faça:
```

```
    1.1. Exiba na tela a mensagem "Parabéns!".
```

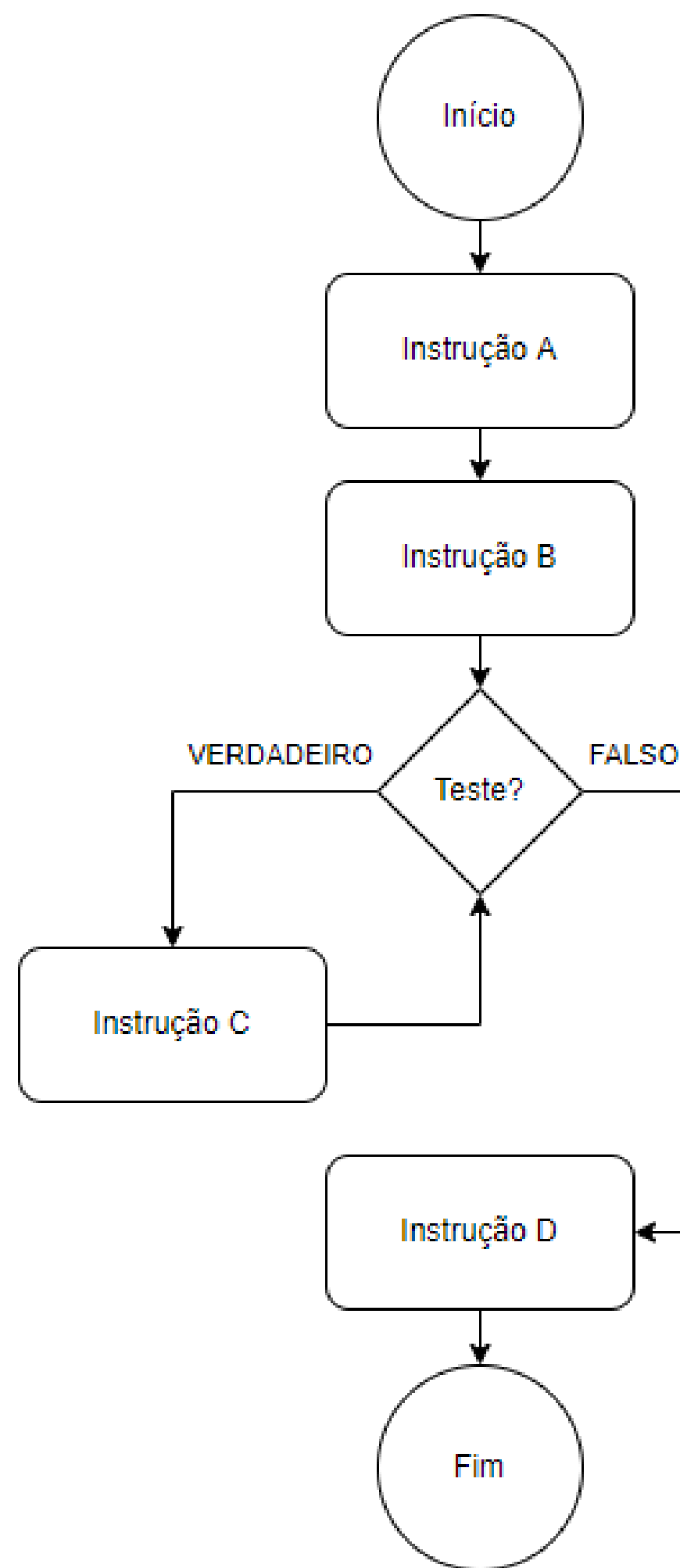
```
2. Senão, faça:
```

```
    2.1. Exiba na tela a mensagem "Continue estudando!".
```

```
3. Exiba na tela a mensagem "Sua nota foi ", seguida pelo valor de nota.
```

```
4. Encerre o procedimento.
```

Estruturas de repetição



```
PROCEDIMENTO progressao_geometrica(base)
```

```
-----  
# Entrada: parâmetro base é um número inteiro.
```

```
# Saída: nenhuma.  
-----
```

```
1. Se  $base < 0$ , encerre o procedimento.
```

```
2. Considere uma variável valor igual a 1, e uma total_elementos igual a 0.
```

```
3. Enquanto  $valor < 1000$ , faça:
```

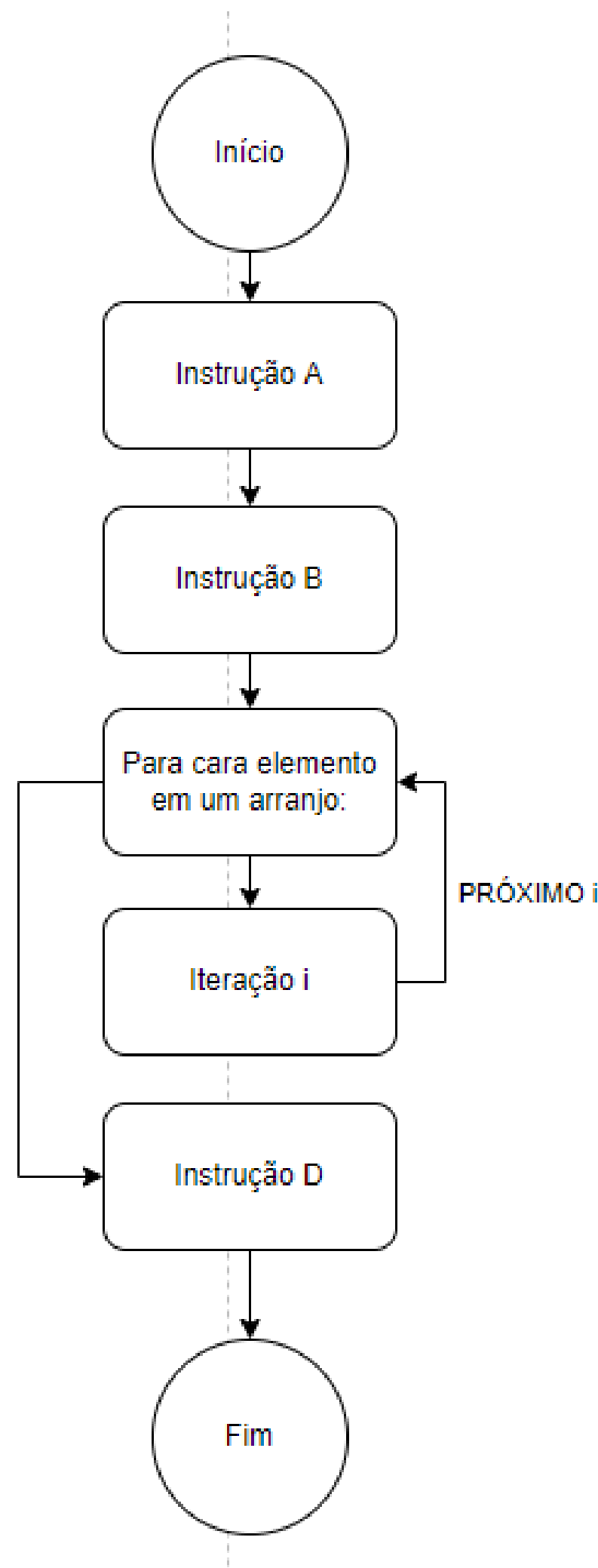
```
    3.1. Exiba valor na tela.
```

```
    3.2. Acrescente 1 a total_elementos.
```

```
    3.3. Atribua para valor o resultado de  $valor * base$ .
```

```
4. Encerre o procedimento, retornando total_elementos como saída.
```

Estruturas de repetição



```
PROCEDIMENTO busca_linear(A, n, x)
```

```
-----
```

```
# Entradas:
```

```
# - A: um arranjo.
```

```
# - n: o número de elementos em A no qual procurar.
```

```
# - x: o valor que está sendo procurado.
```

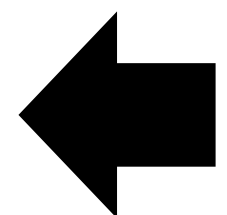
```
# Saída: um índice i para o qual  $A[i] = x$  ou o valor -1,  
# que indica que o valor de x não foi encontrado no arranjo.
```

```
-----
```

```
1. Para  $i = 0$  até n, faça:
```

```
    1.1. Se  $A[i] = x$ , então retorne o valor de i como saída.
```

```
2. Retorne -1 como saída.
```



Noções básicas de OO

Por que usar OO?

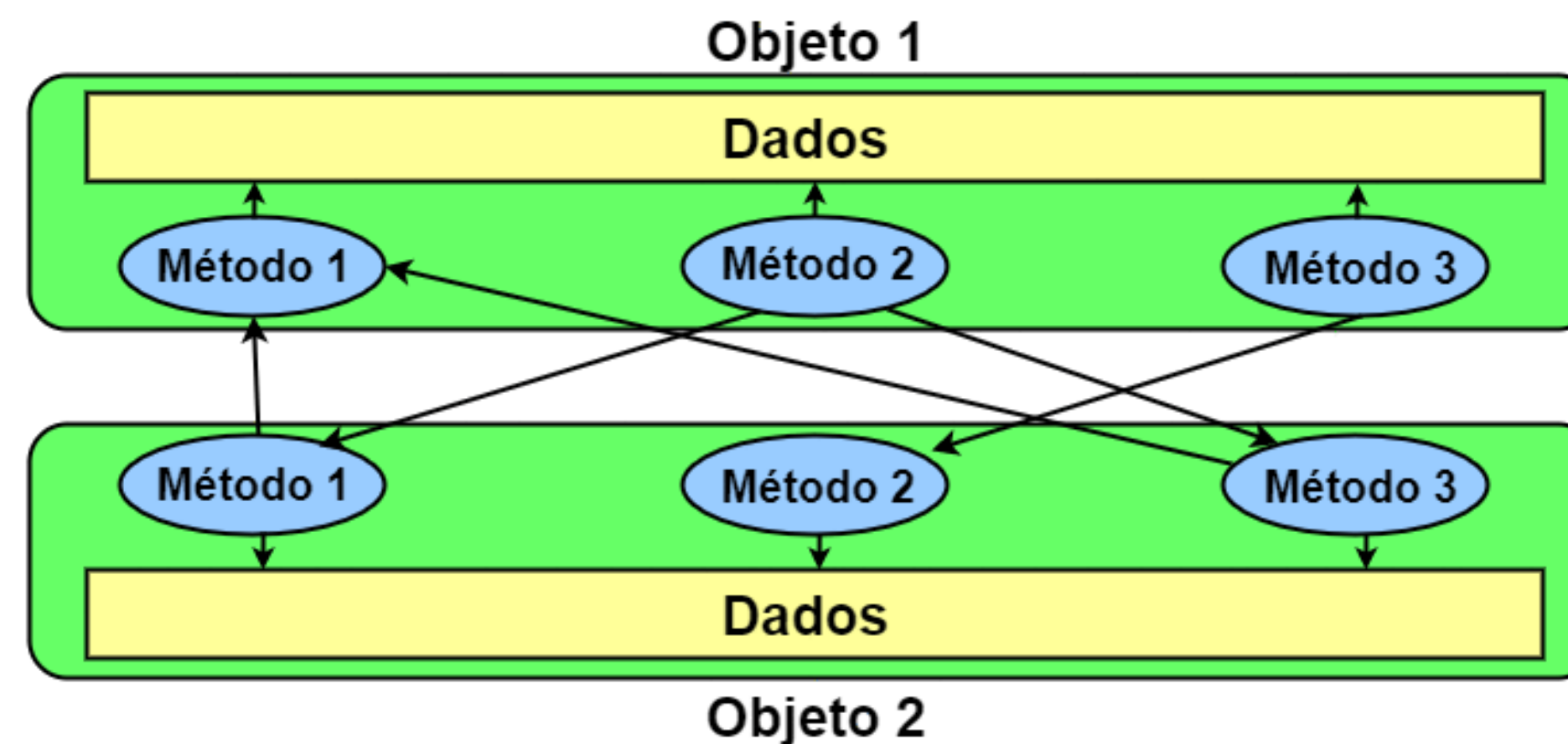
- Segundo o Paradigma Procedural, é possível representar todo e qualquer processo do mundo real a partir da utilização de **apenas** três estruturas básicas:
 - Sequência: Os passos devem ser executados um após o outro, linearmente. Ou seja, o programa seria uma sequência finita de passos. Em uma unidade 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.

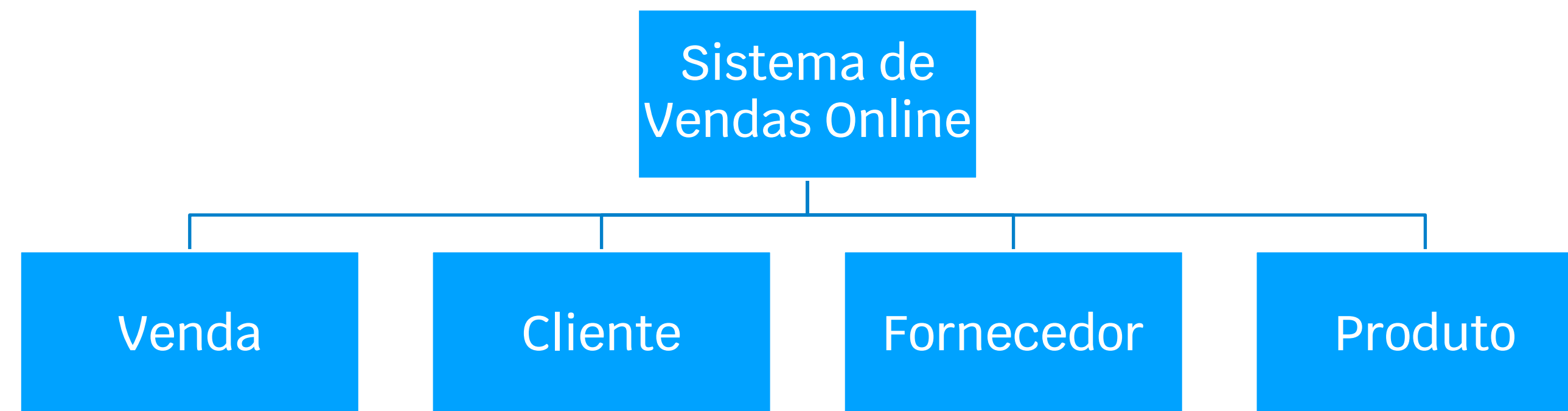
- Essa mesma ideia se aplica na OO. No caso, a complexidade que desejamos esconder é a de implementação de alguma necessidade. Com o encapsulamento, podemos esconder a forma como algo foi feito, dando a quem precisa apenas o resultado gerado.
- Uma vantagem deste princípio é que as mudanças se tornam 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

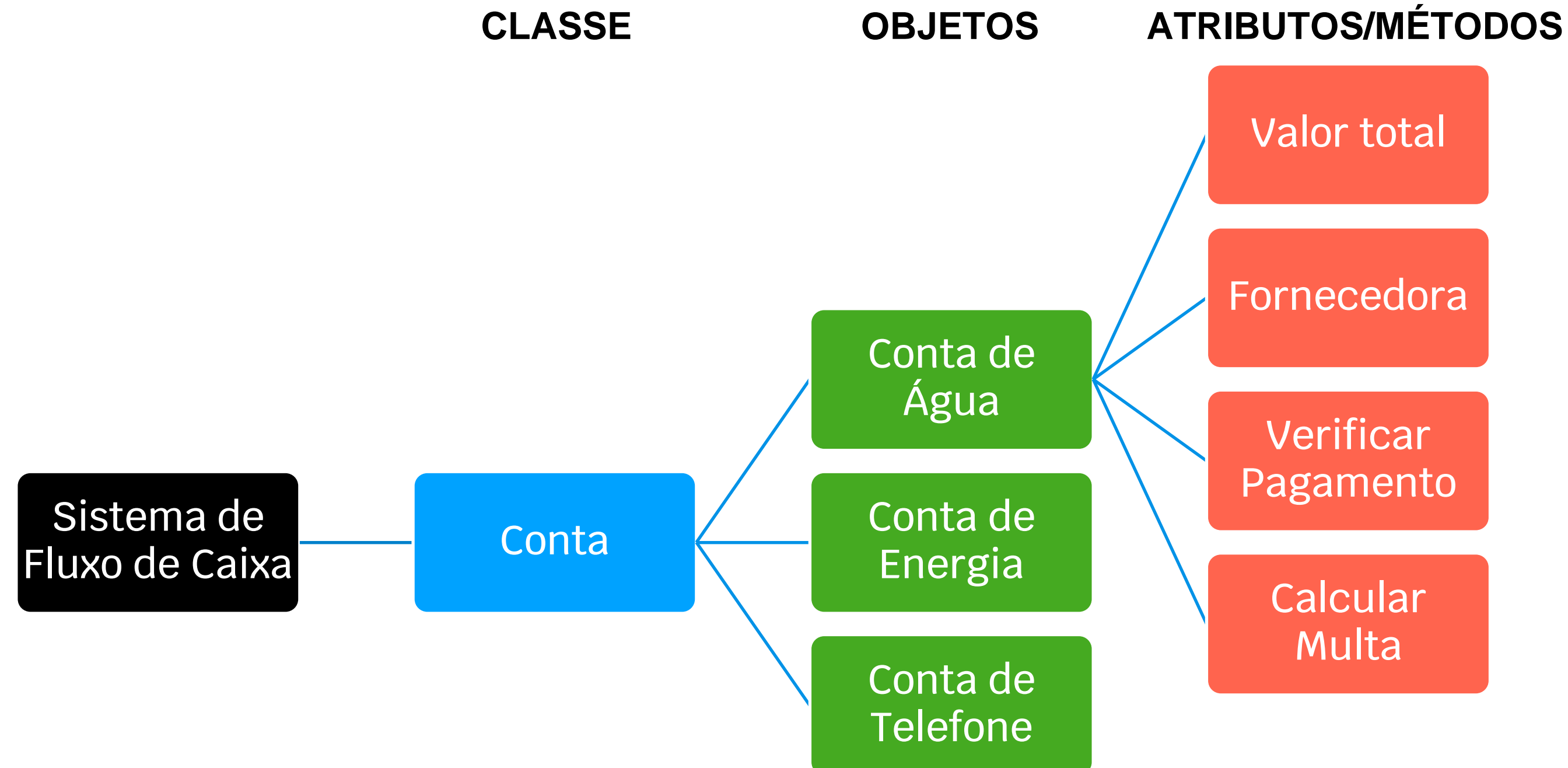
- Há também a possibilidade de passarmos somente tipos primitivos, entretanto, isto remete à programação procedural e deve ser desencorajado. Via de regra, se você passa muitos parâmetros separados, talvez eles pudessem representar algum conceito em conjunto. Neste caso, valeria a pena avaliarmos se 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).



Git

Introdução

- Talvez você reconheça esse nome por causa de sites como *github* ou *gitlab*. **Git** é um sistema *open-source* de controle de versionamento.
- Versionar projetos é uma prática essencial no mundo profissional e, em particular, na área de tecnologia, para manter um histórico de modificações deles, ou poder reverter alguma modificação que possa ter comprometido o projeto inteiro, dentre outras funcionalidades que serão aprendidas na prática.
- Os projetos são normalmente armazenados em **repositórios**.
- Vamos aqui falar de alguns conceitos principais sobre como funciona o sistema, independente da plataforma que vamos utilizar (p.ex., *github*). Em seguida vamos aplicar alguns desses conceitos na prática.

Para mais informações...

- O tempo que temos em aula não é suficiente para conseguirmos discutir todos os detalhes por trás dessa tecnologia. Portanto, seguem abaixo algumas sugestões de conteúdos extras para estudarem:
 - <http://git-scm.com/book/en/Getting-Started-About-Version-Control>
 - <http://git-scm.com/book/en/Getting-Started-Git-Basics>
 - <http://learngitbranching.js.org/>
 - <http://try.github.io/levels/1/challenges/1>

O que é Controle de Versões?

- Controle de Versões é um sistema de grava mudanças aplicadas em um arquivo ou um conjunto de arquivos ao longo do tempo, para que o usuário possa relembrar ou recuperar depois.
- Na área de tecnologia o controle de versões já é algo consolidado, uma vez que inúmeras alterações são aplicadas em um software durante a sua implementação e manutenção. No entanto, adotar um sistema de controle de versões (VCS, da sigla em inglês) é algo recomendado para qualquer área.
- Quando se quer adotar um VCS, normalmente o primeiro passo é trabalhar com um controle local, fazendo cópias dos arquivos e os renomeando com algum padrão (p.ex., incluindo a data ao final do nome do arquivo).
- O Git veio para otimizar esse controle de versões, permitindo inúmeras alterações que seriam muito complicadas se fossem feitas manualmente.

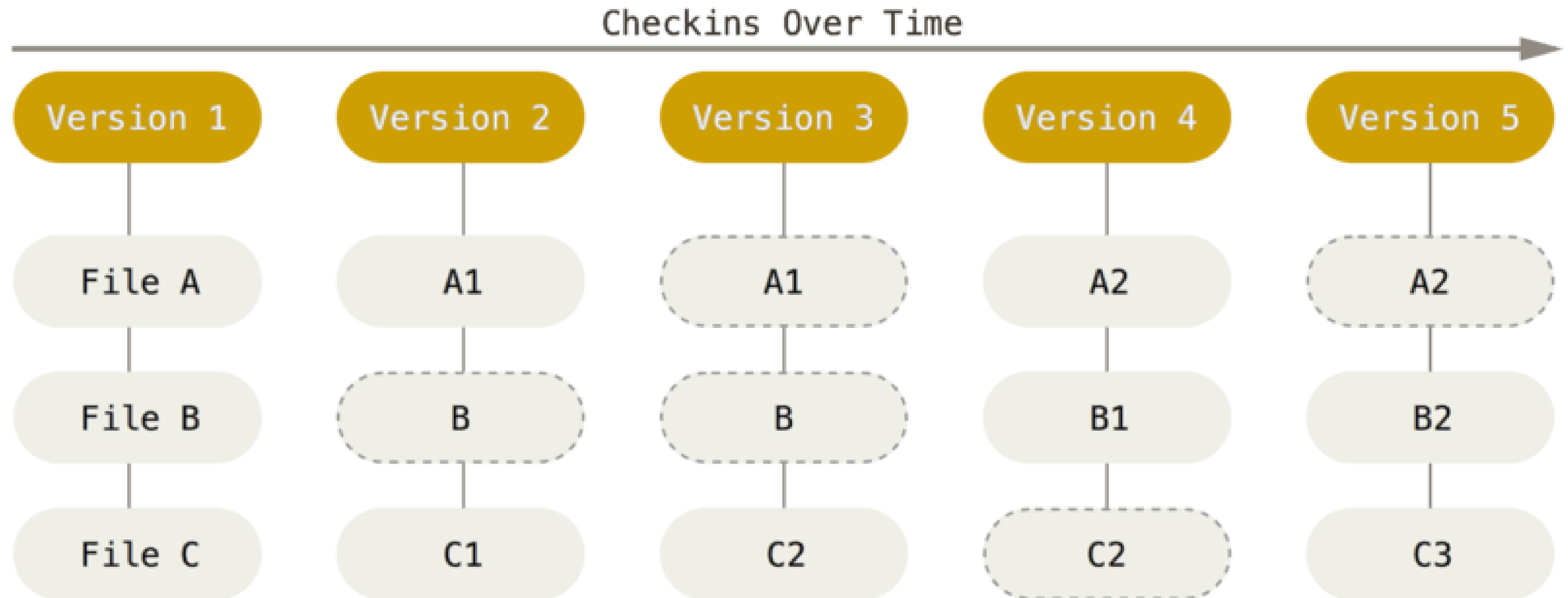
Uma breve história do Git

- A criação do Git está atrelada à criação do Linux, quando o time responsável utilizou uma solução de controle de versão que, eventualmente, tornou-se paga.
- A comunidade Linux, portanto, começou a planejar um sistema que aproveitasse algumas das características da solução anterior, porém evoluindo diversos aspectos.
- A comunidade tinha os seguintes objetivos para o novo sistema:
 - Velocidade
 - Design simples
 - Suporte forte para desenvolvimento não-linear, com milhares de atividades sendo realizadas em paralelo
 - Completamente distribuído, permitindo o acesso de qualquer lugar
 - Eficiente no suporte a grandes projetos

O que é Git?

- **Git** funciona pensando que os dados de um repositório compõem uma série de “fotografias” de um sistema de arquivos em miniatura.
- No Git, a cada vez que você aplica uma alteração (ou *commit*), ou salva o estado do seu projeto, o Git basicamente tira uma “foto” de como os arquivos do repositório estão naquele momento e então ele armazena uma referência a essa foto.
- Para ser eficiente, se os arquivos não foram alterados, o Git não altera os arquivos novamente, apenas um link para a última versão do arquivo armazenada.
- Sendo assim, o Git trabalha os dados como um **fluxo de fotografias**.

O que é Git?



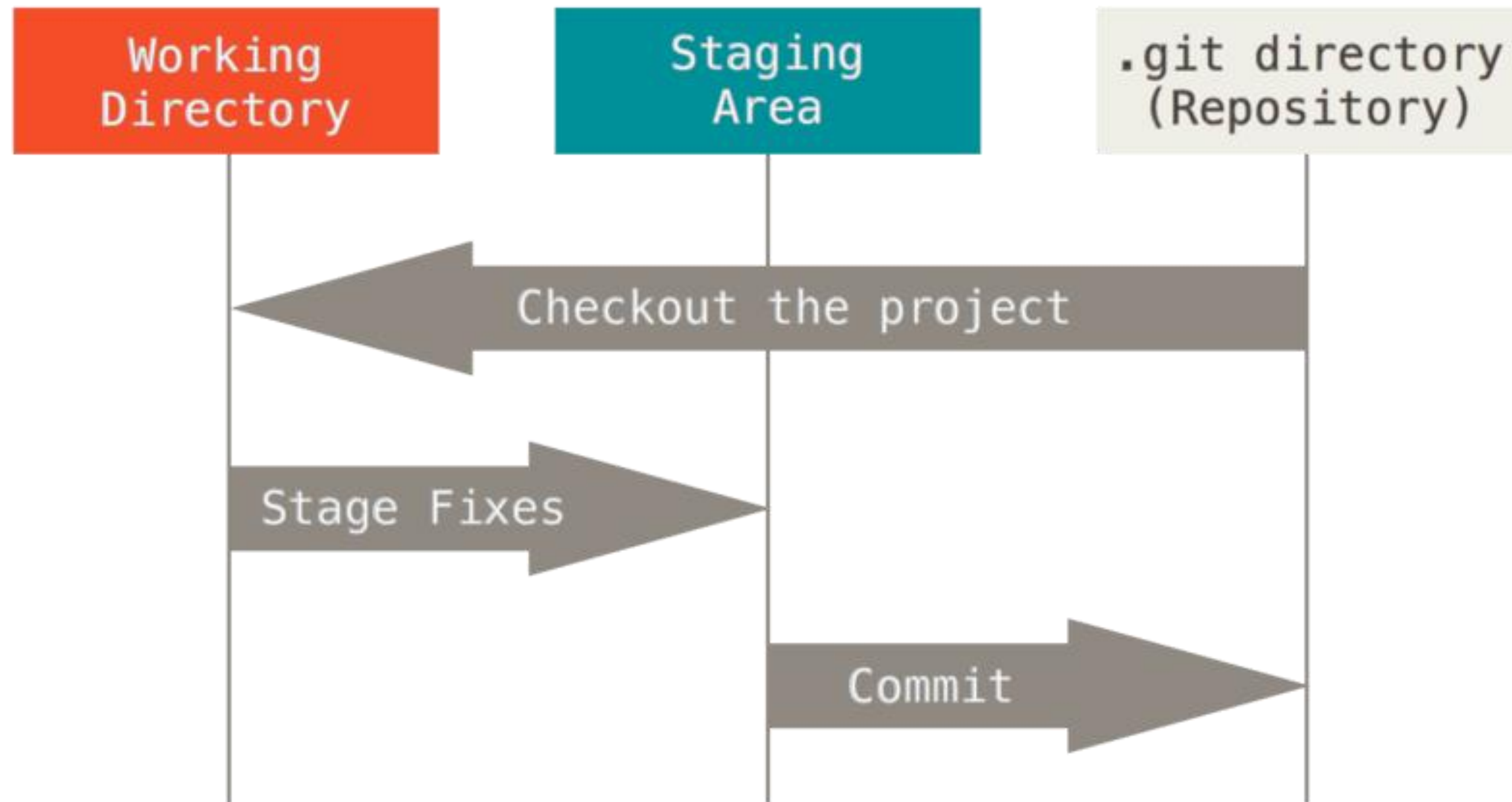
O que é Git?

- A maior parte das operações no Git precisa apenas de arquivos e recursos locais para operarem, e normalmente nenhuma informação é necessária de outro computador na rede. Isso fornece uma velocidade de operação que outros VCS não possuem. Como cada usuário possui todo o histórico do projeto no computador, a maioria das operações aparenta ser quase instantânea.
- Para todos os efeitos, na prática, o Git normalmente só acrescenta informações ao seu banco de dados, nunca removendo informações. É muito difícil, e não recomendado, gerar operações que removam informações, já que essas operações podem afetar o histórico do seu projeto.

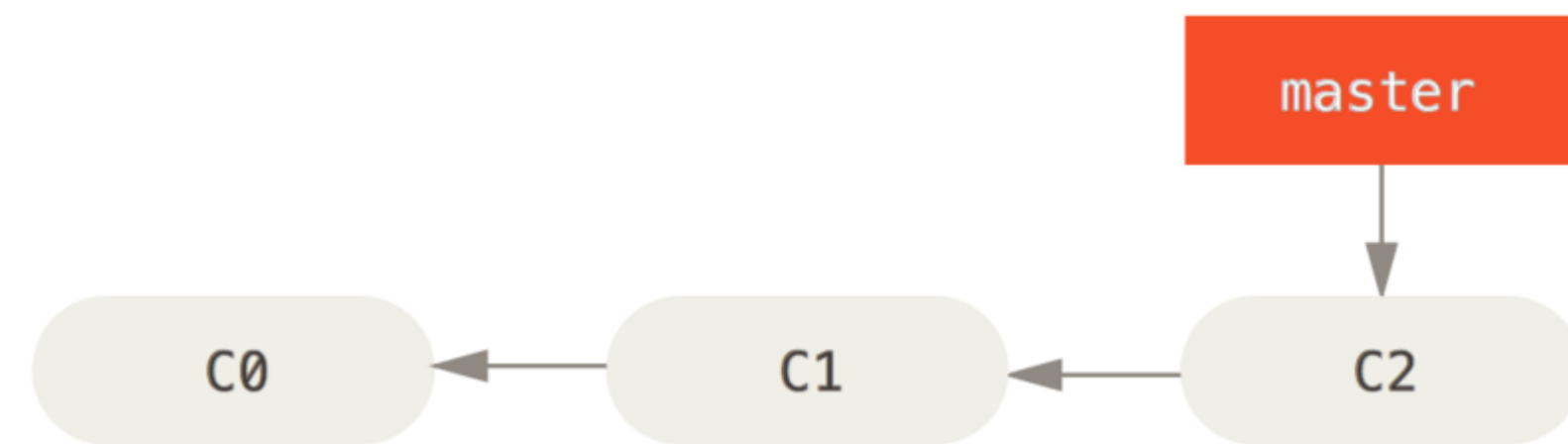
Os três estados

- O Git tem três estados principais nos quais os arquivos de um repositório podem se encontrar: **modificado**, **preparado** e **“commitado”**:
 - **Commitado** significa que os dados estão armazenados de forma segura em seu banco de dados local;
 - **Modificado** significa que você alterou o arquivo, mas ainda não fez o commit no banco de dados;
 - **Preparado** significa que você marcou a versão atual de um arquivo modificado para fazer parte do seu próximo commit.
- Isso leva a três seções principais de um projeto Git: o diretório Git, o diretório de trabalho e área de preparo.

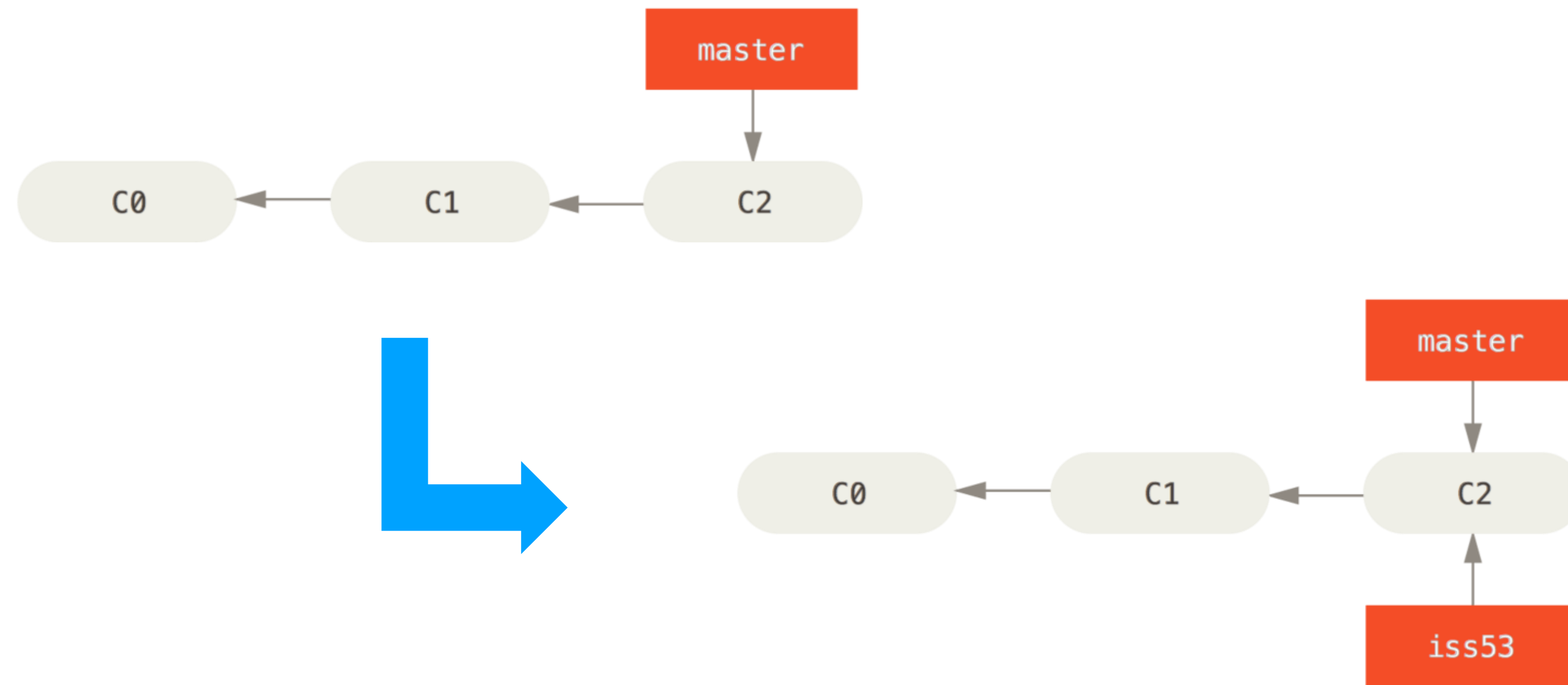
Os três estados



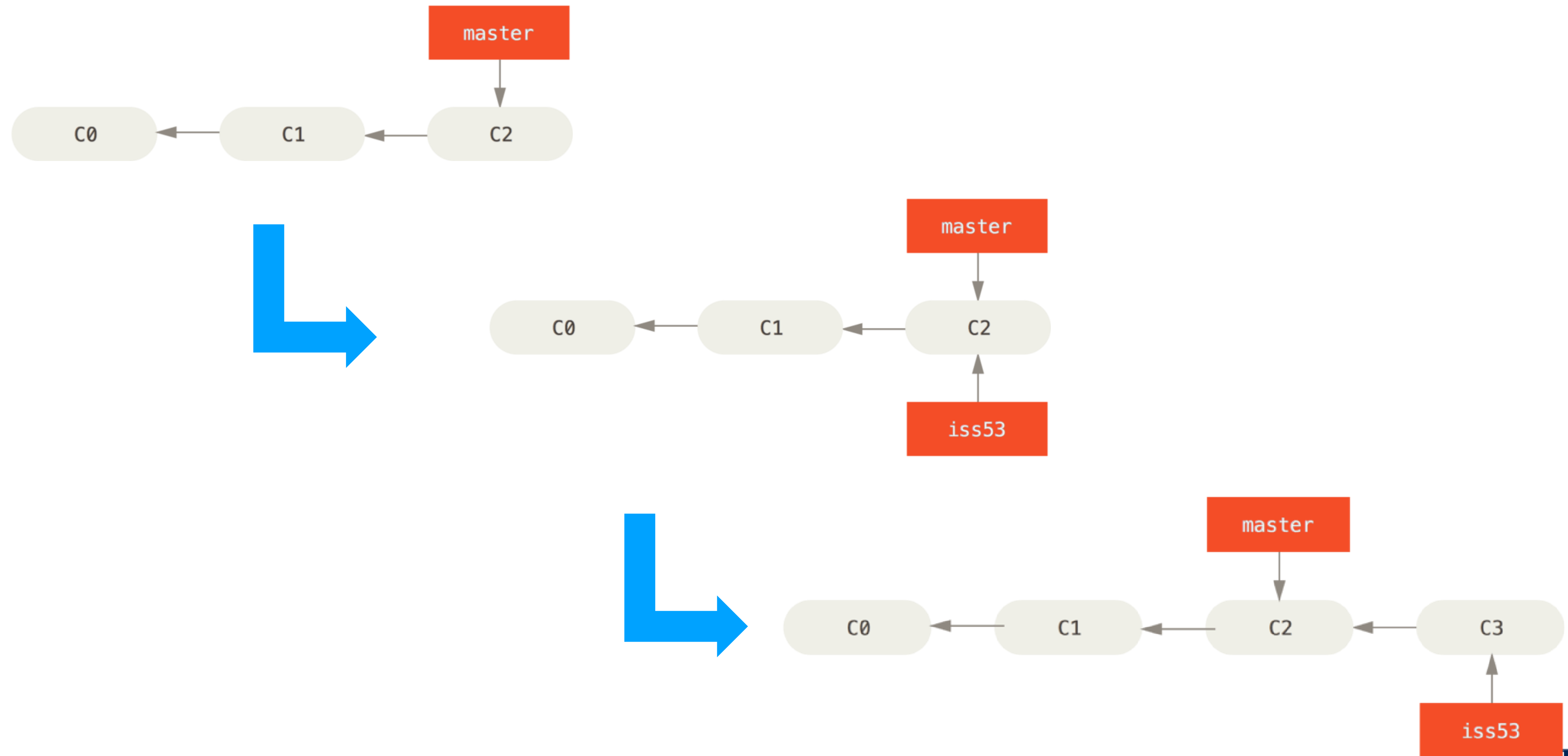
Branches no Git



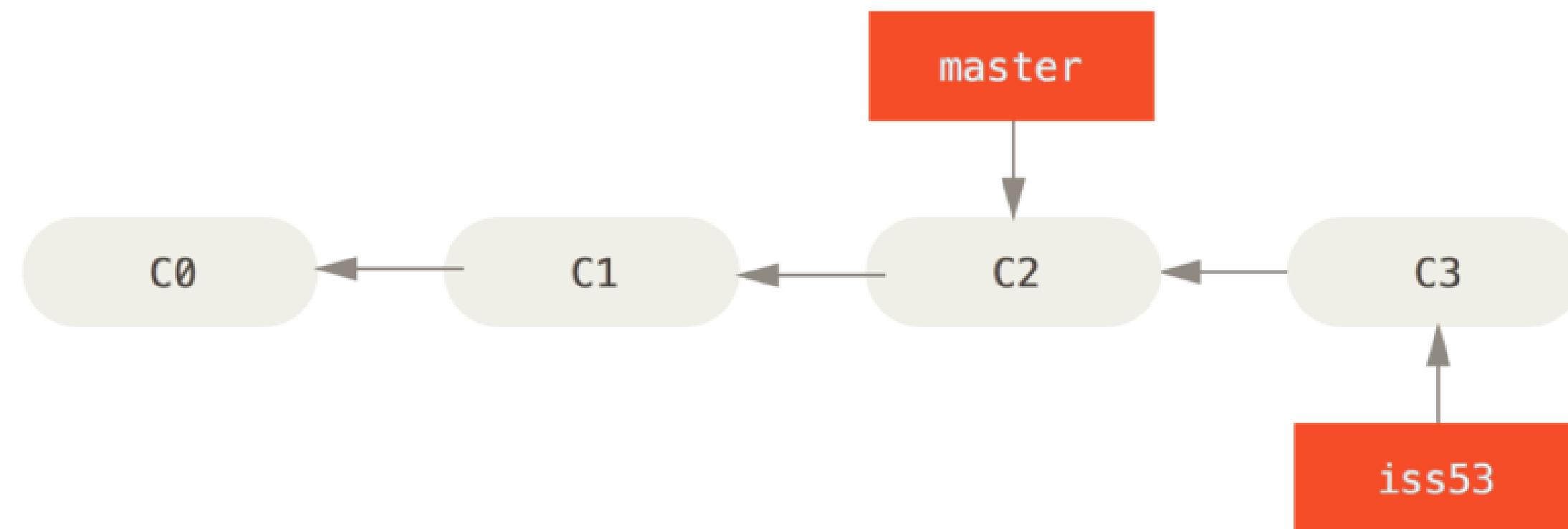
Branches no Git



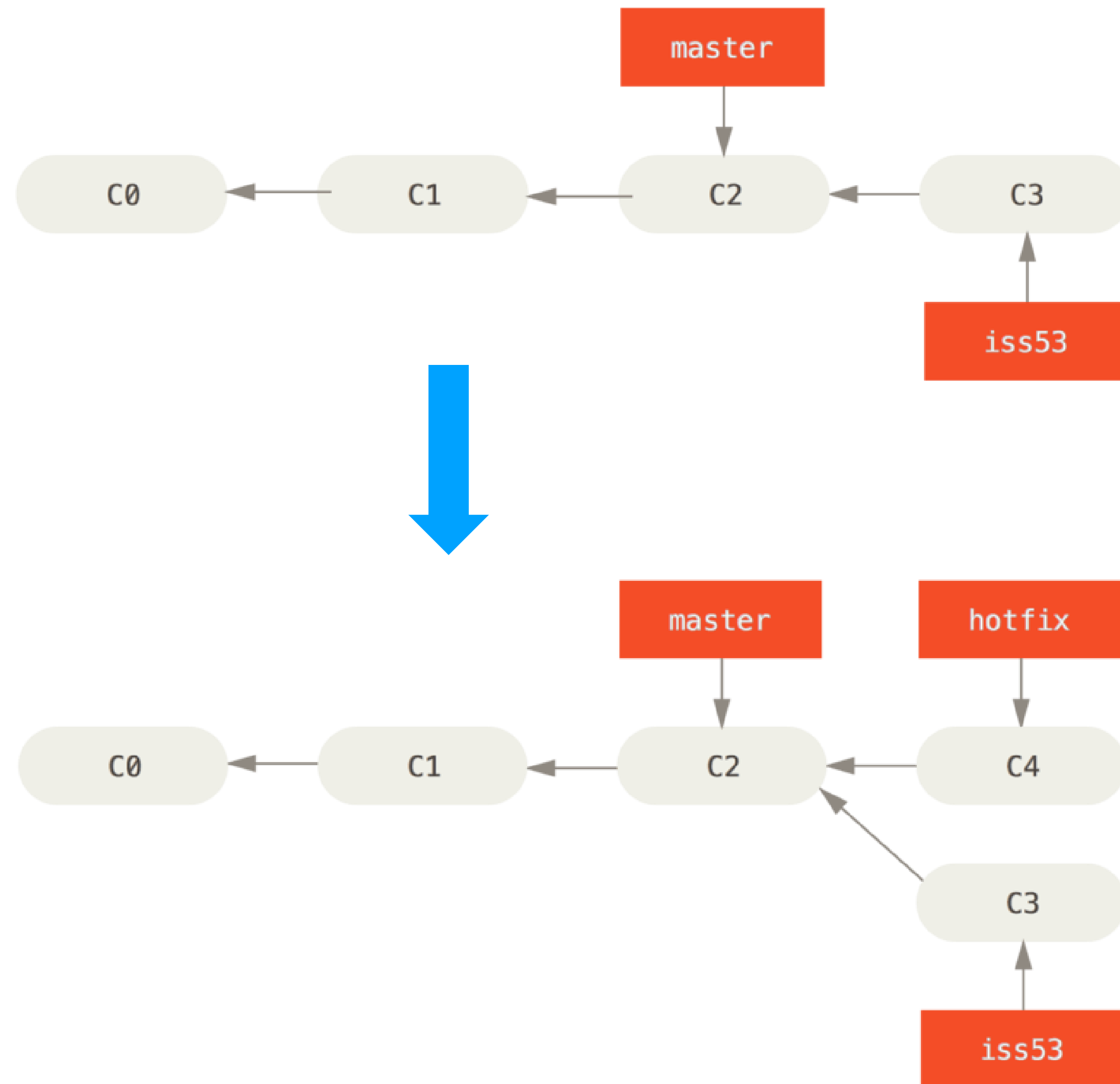
Branches no Git



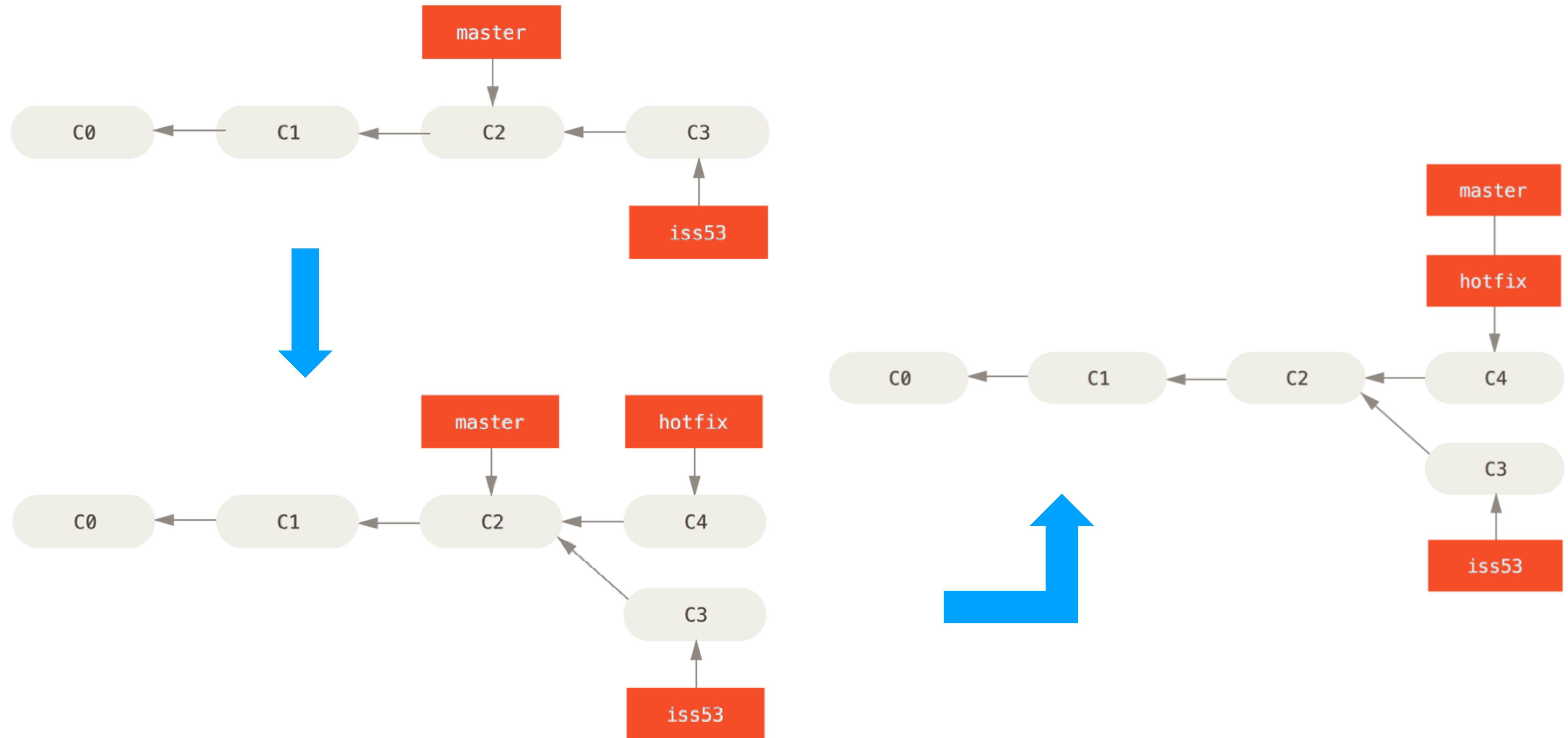
Branches no Git



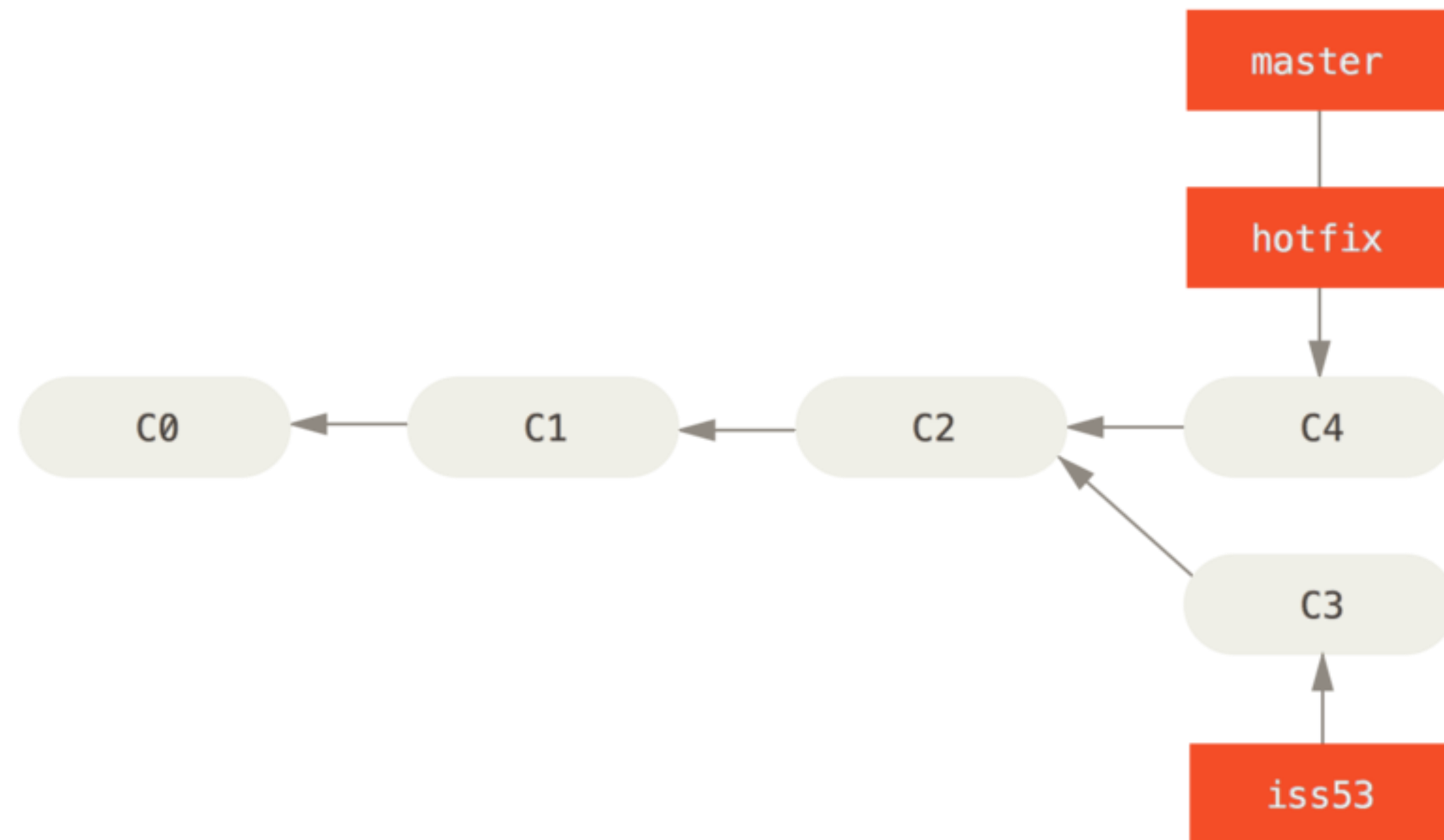
Branches no Git



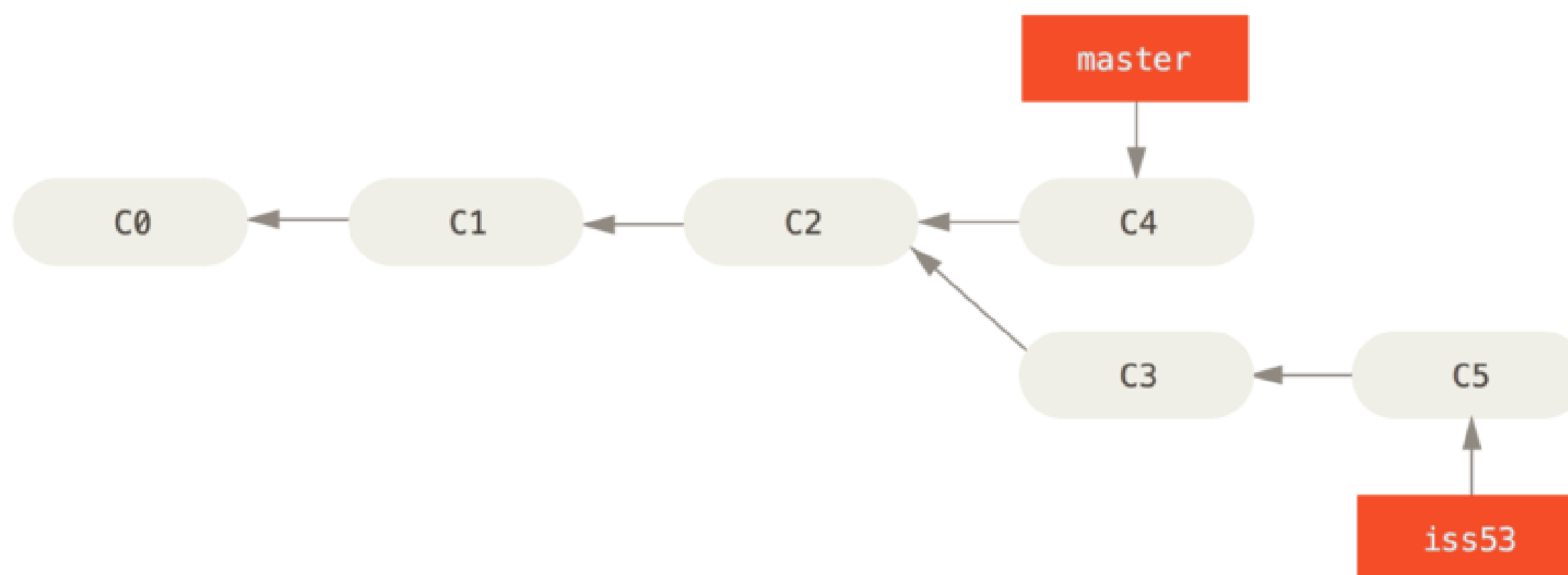
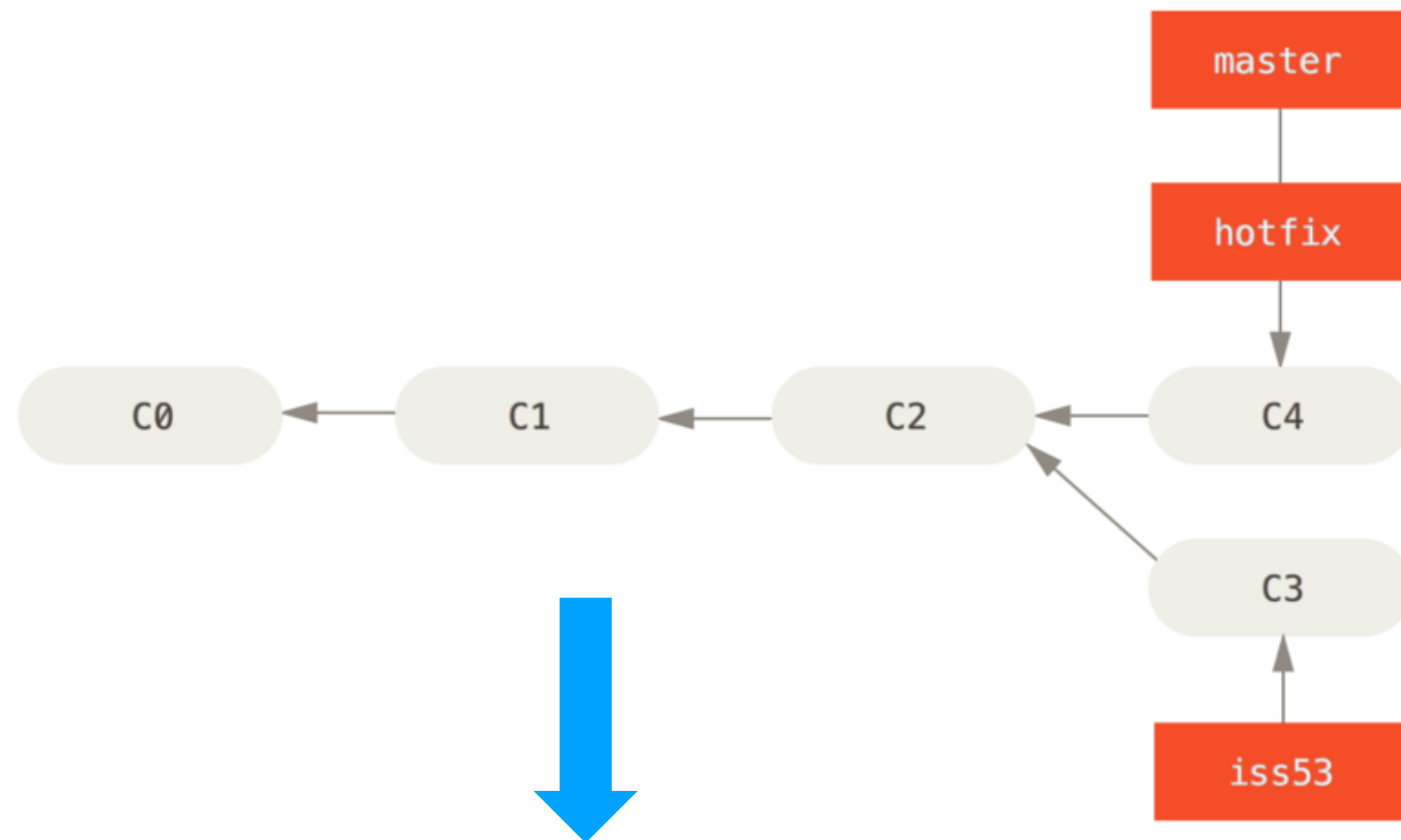
Branches no Git



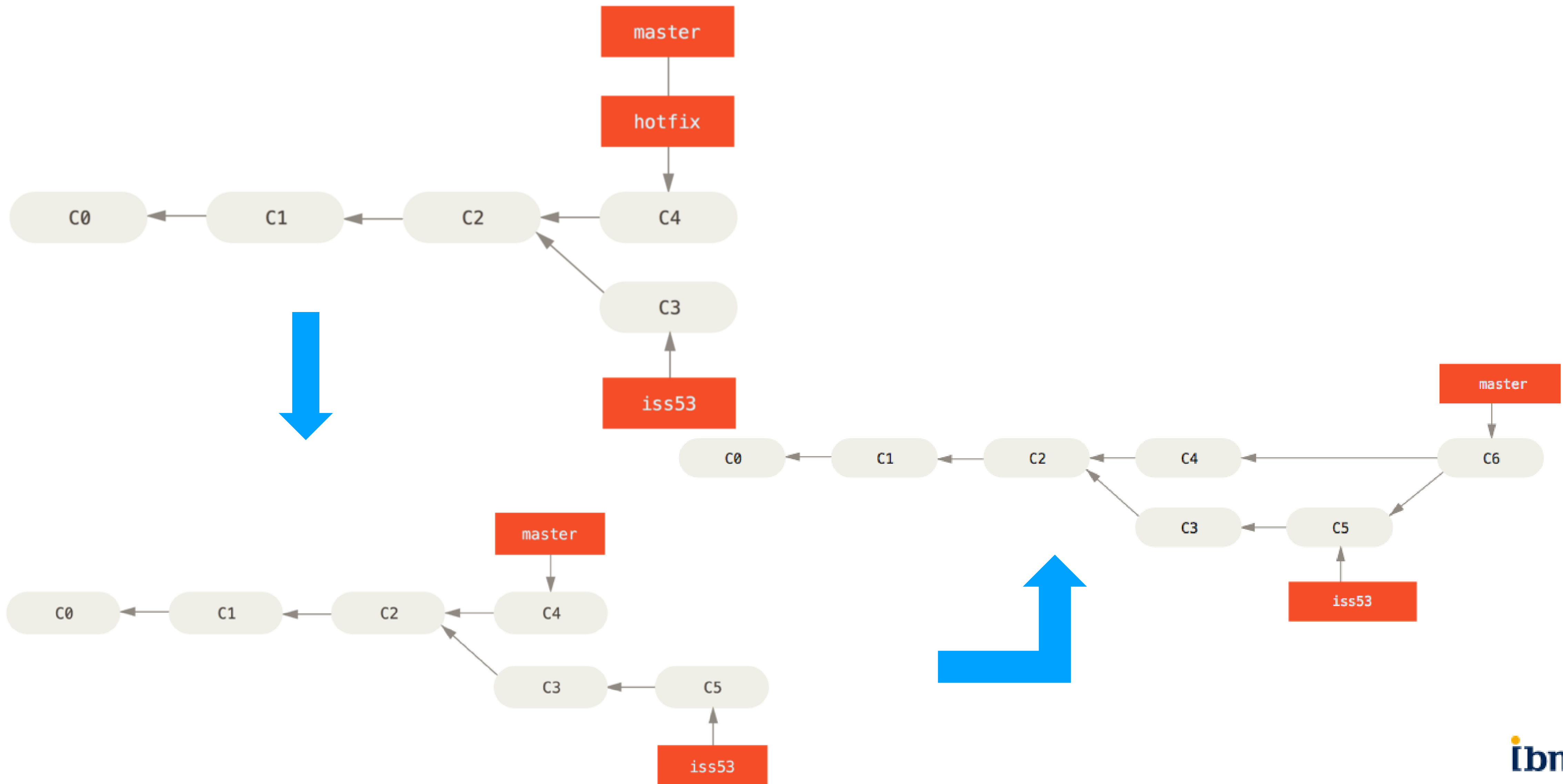
Branches no Git



Branches no Git



Branches no Git

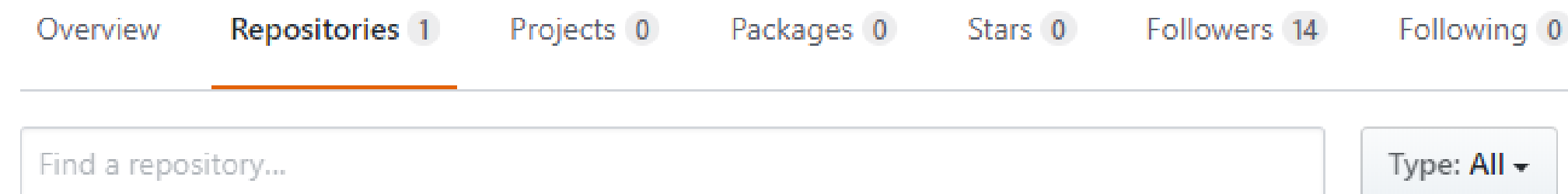


Iniciando o uso de Git com GitHub

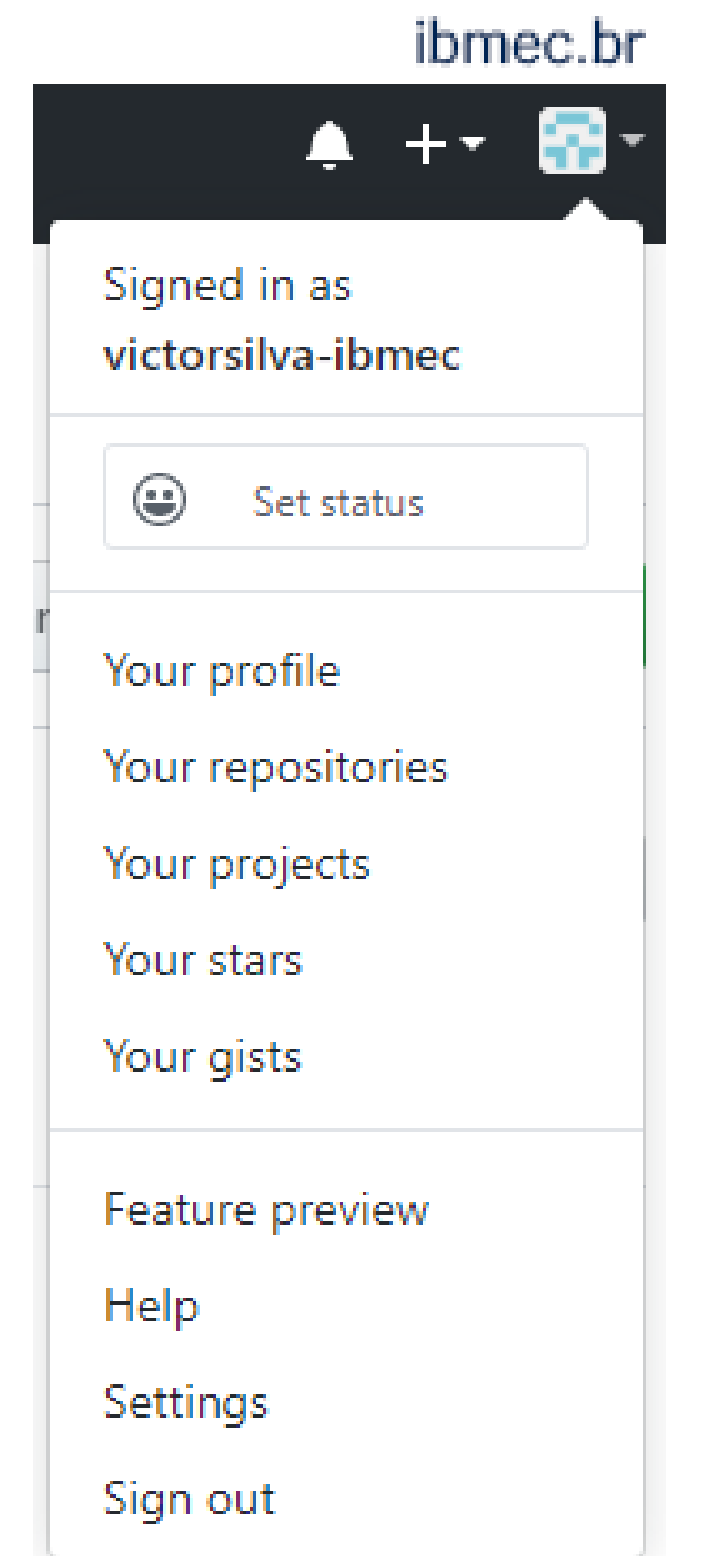
- Antes de mais nada, precisamos instalar o sistema Git na nossa máquina.
- <https://git-scm.com/downloads>
- O GitHub (www.github.com) é uma plataforma de versionamento que utiliza Git como base.
- Não é o nosso objetivo ensinar a plataforma a fundo, mas vamos utilizar algumas funcionalidades. Caso tenha interesse em se aprofundar no assunto, recomendo algumas páginas:
 - <https://github.com/culturagovbr/primeiros-passos>;
 - <https://help.github.com/en>;
 - https://rogerdudler.github.io/git-guide/index.pt_BR.html.
- Antes de começar a configurar, crie uma conta no site.

Algumas tarefas no GitHub

- Criando um novo repositório:
 - No canto superior direito, clique no ícone do seu usuário e, em seguida, em **Your Profile**;
 - Na nova janela, clique em **Repositories** e em **New**;

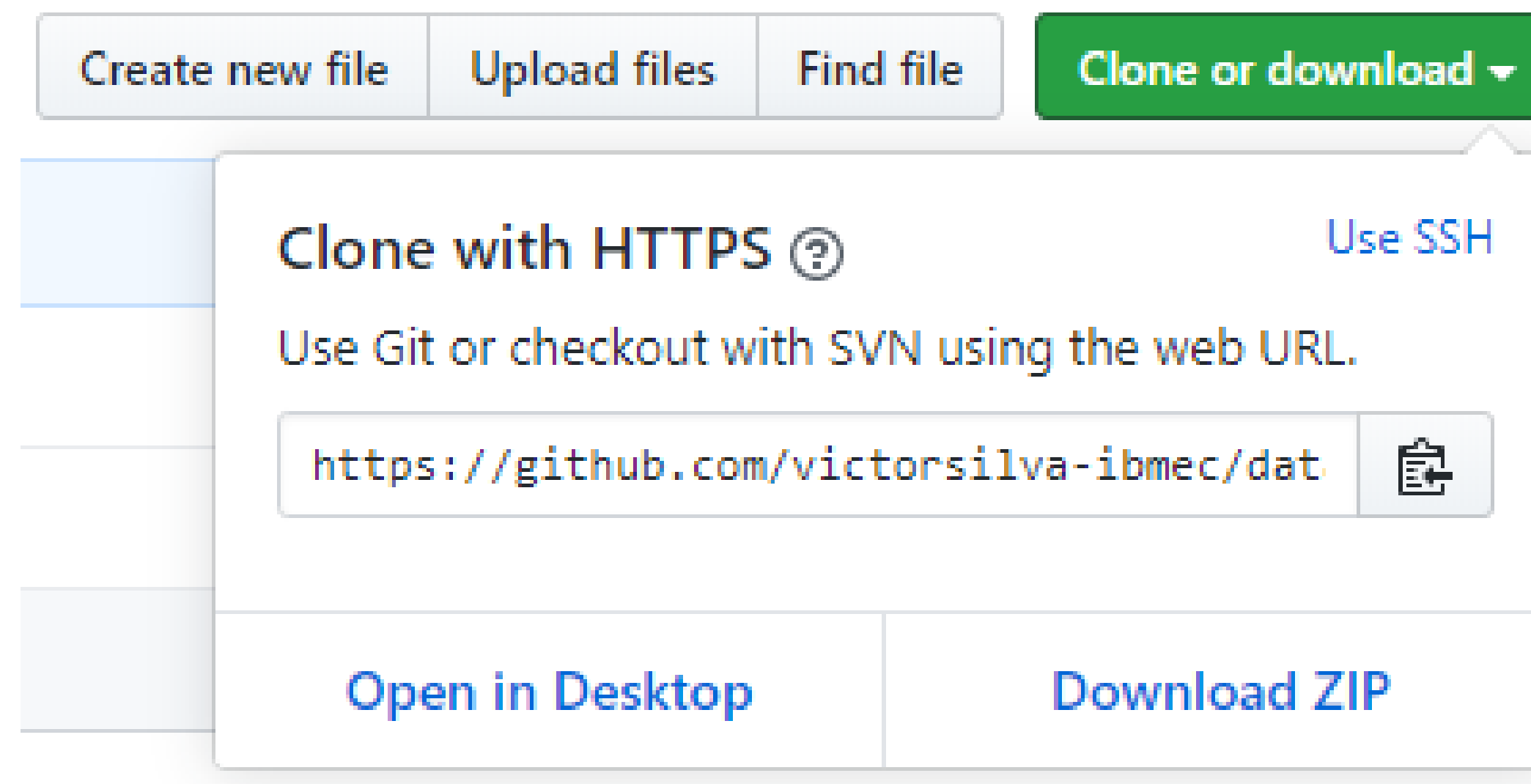


- Defina um nome (evite espaços e números), insira uma descrição se desejar e marque a opção **Public**, para que ele possa ser compartilhado. Por fim, marque a caixa **Initialize this repository with a README**;
- Clique em **Create repository**.

This screenshot shows the 'Create new repository' form on GitHub. The 'Owner' is set to 'victorsilva-ibmec' and the 'Repository name' is 'data-mining'. A note states: 'Great repository names are short and memorable. Need inspiration? How about upgraded-octo-bassoon?'. The 'Description (optional)' field contains 'Repositório para curso de Data Mining com Python'. The 'Visibility' section has 'Public' selected, with the subtext 'Anyone can see this repository. You choose who can commit.' The 'Private' option is also visible. A note says 'Skip this step if you're importing an existing repository.' The 'Initialize this repository with a README' checkbox is checked, with the subtext 'This will let you immediately clone the repository to your computer.' There are also dropdowns for 'Add .gitignore: None' and 'Add a license: None', and a 'Create repository' button at the bottom.

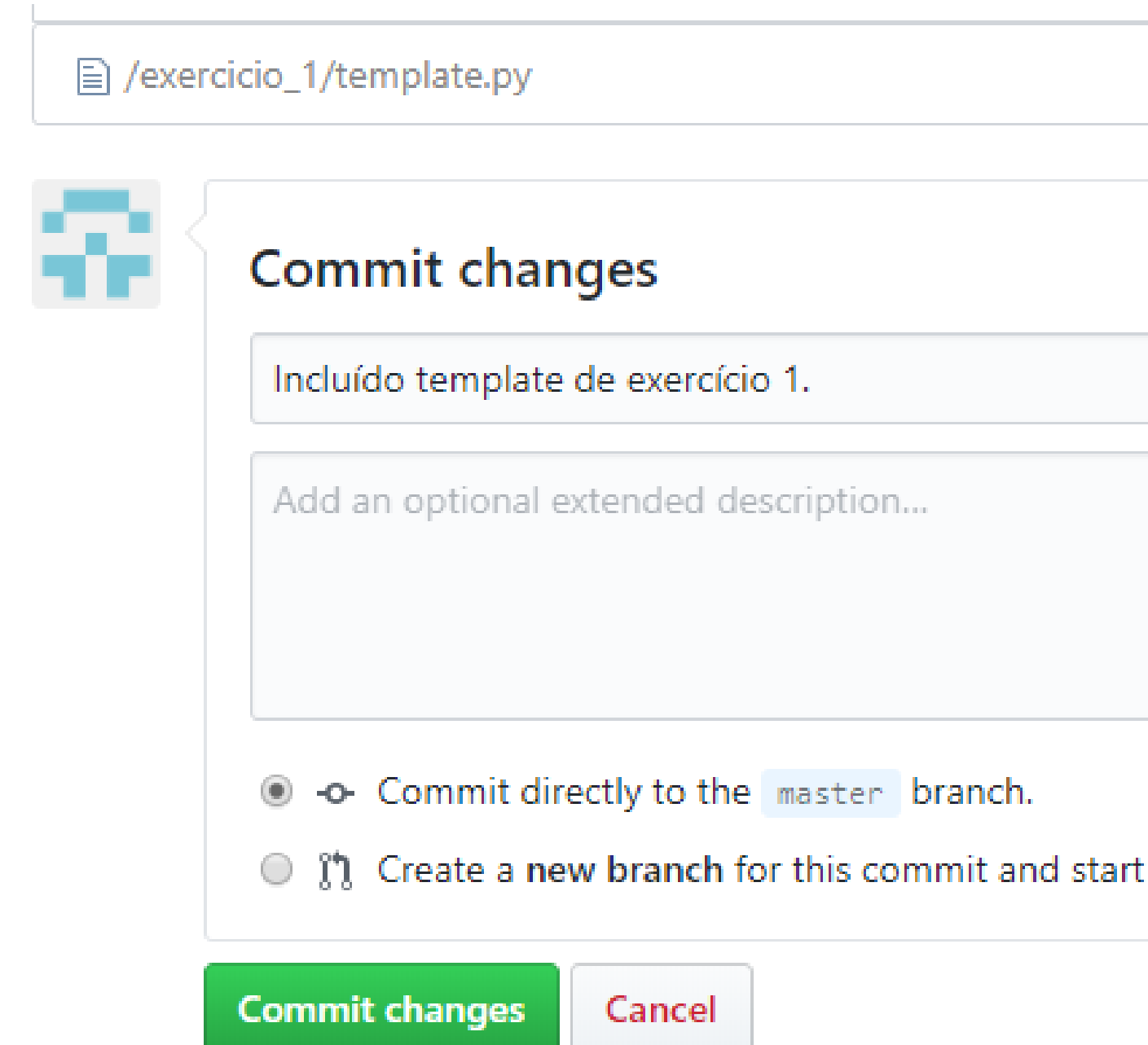
Algumas tarefas no GitHub

- Baixando manualmente um repositório para a máquina:
 - Na tela do seu repositório, clique em **Clone or download**;
 - Em seguida, clique em **Download ZIP**;
 - Com o arquivo .zip baixado, descompacte-o na sua pasta de projeto, substituindo arquivos antigos se necessário.



Algumas tarefas no GitHub

- **Fazendo um novo commit no seu repositório:**
 - Commits são alterações feitas no repositório. Podem conter um único arquivo ou vários. Caso um arquivo commitado já exista, o GitHub vai fazer um controle de versão, comparando as alterações entre a versão anterior e a que foi commitada;
 - Na tela do seu repositório, clique em **Upload files**;
 - Arraste para a tela os arquivos que deseja incluir;
 - Insira uma breve descrição do que está sendo commitado;
 - Deixe marcada a opção **Commit directly to the master branch**;
 - Clique em **Commit changes**.



/exercicio_1/template.py

Commit changes

Incluído template de exercício 1.

Add an optional extended description...

☒ Commit directly to the `master` branch.

☐ Create a new branch for this commit and start

Commit changes Cancel

Algumas tarefas no GitHub

- Submetendo um trabalho para revisão:
 - Um **pull request** é o ato de submeter para aprovação as alterações ou inserções de código de um ou mais arquivos. A pessoa que abre um **pull request** sinaliza que gostaria de uma aprovação do conteúdo antes de ele ser, de fato, incorporado ao repositório;
 - No repositório desejado, clique na pasta em que você deseja incluir ou atualizar os arquivos;
 - Clique em **Upload files**;
 - Arraste para a tela o(s) arquivo(s) com a sua atualização, e na descrição explique o que está sendo feito. Em seguida, clique em **Commit changes**;
 - Na nova janela, insira um comentário e depois clique em **Create pull request**.

Alguns comandos do Git

- Apesar do Github fornecer recursos para operar com Git direto pelo navegador, esses recursos são limitados. Por exemplo, fazer checkout de um novo branch apenas pelo navegador pode ser bem complicado.
- Uma forma mais usual de se usar o Git é através de programas específicos para o computador.
- É bem comum utilizar os comandos do Git por linha de comando, porém existem bons programas para uso do Git através de uma interface gráfica, como o [Sourcetree](#).
- No slide a seguir serão apresentados alguns comandos comuns para o uso do Git pela linha de comando. Eles podem ser executados pelo Git Bash (programa instalado junto com o Git), ou pelo terminal.

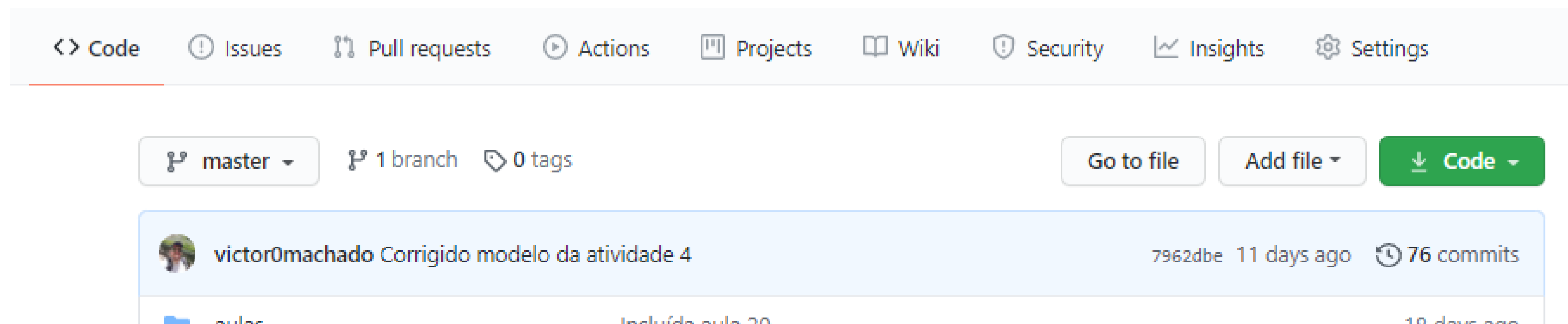
Alguns comandos do Git

- Uma observação, para facilitar as operações, é trabalhar sempre na raiz do repositório.
- Com o terminal na raiz do repositório, insira os seguintes comandos para obter os efeitos apresentados:

| Comando | Efeito |
|--------------------------------|--|
| git init | Inicia um repositório Git no diretório em questão |
| git status | Indica os status de arquivos modificados, adicionados ou removidos, além de arquivos preparados (staged) |
| git add <arquivo> | Prepara o arquivo mencionado |
| git add -u | Prepara todos os arquivos modificados (porém não faz nada com arquivos novos) |
| git add . | Prepara todos os arquivos (incluindo arquivos novos) |
| git commit -m "Mensagem" | Faz um commit dos arquivos preparados, incluindo a mensagem de commit definida |
| git restore <arquivo> | Desfaz modificações do arquivo que não foi preparado |
| git restore --staged <arquivo> | Desfaz a preparação do arquivo (porém mantém modificações) |
| git pull | No branch escolhido, atualiza as informações com o repositório remoto |
| git branch | Lista todos os branches armazenados localmente |
| git branch --show-current | Lista o branch atual |
| git branch -m <novo_nome> | Renomeia o branch atual (cuidado ao fazer isso para branches que já estão no repositório remoto!) |
| git checkout <branch> | Dá checkout no branch mencionado |
| git checkout -b <branch> | Cria um novo branch, com o nome mencionado, e dá checkout nele |
| git push | Envia os commits realizados localmente para o branch remoto |

GitHub pages

- Uma boa forma de manter o seu portfólio sempre atualizado é com uma página pessoal, na qual você inclui seu currículo, projetos realizados, trabalhos, interesses pessoais e profissionais, e outras informações que achar pertinente.
- O Github possui uma forma muito simples de se criar um repositório que também serve como página pessoal.
- Para isso, crie um repositório normal, vá na página desse repositório e clique em “Settings”.



GitHub pages

- Nas configurações, desça a página até encontrar a seção “GitHub Pages”.

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Source
GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. [Learn more.](#)

None ▾ Save

Theme Chooser
Select a theme to publish your site with a Jekyll theme using the gh-pages branch. [Learn more.](#)

Choose a theme

- No campo “Source”, indique o branch que você quer que seja a sua página principal (usualmente é o branch master). Se quiser, escolha um tema da lista de temas gratuitos disponíveis e, em seguida, clique em “Save”.

GitHub pages

- A página terá uma atualização que mostrará a URL do site, além de incluir um campo no qual você pode inserir um domínio customizado, caso o tenha.

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is ready to be published at <https://victor0machado.github.io/2020.2-logprog/>.

Source

Your GitHub Pages site is currently being built from the master branch. [Learn more.](#)

Branch: master ▾

/ (root) ▾

Save

Theme Chooser

Select a theme to publish your site with a Jekyll theme. [Learn more.](#)

Choose a theme

Custom domain

Custom domains allow you to serve your site from a domain other than victor0machado.github.io. [Learn more.](#)

Save

GitHub pages

- O site funciona como um repositório normal. Todas as páginas devem ser escritas em Markdown, que já vimos ao longo do curso.

```
# Boas-vindas

Vou atualizar essa página com os materiais das disciplinas que leciono no
IBMEC/RJ.

## Disciplinas

* [Algoritmos e Programação de Computadores](/courses/algprog.md)
* [Lógica e Programação de Computadores](/courses/logprog.md)
* [Data Mining com Python](/courses/datamining.md)

## Meus contatos

* E-mail: <victor.silva@professores.ibmec.edu.br>
* [Linkedin](https://www.linkedin.com/in/victormachadodasilva/)
* [Lattes](http://lattes.cnpq.br/1584907276781609)
```

Repositório público do Prof. Victor Machado

Material usado nas minhas disciplinas do
IBMEC/RJ

[View My GitHub Profile](#)

Boas-vindas

Vou atualizar essa página com os materiais das disciplinas que leciono no
IBMEC/RJ.

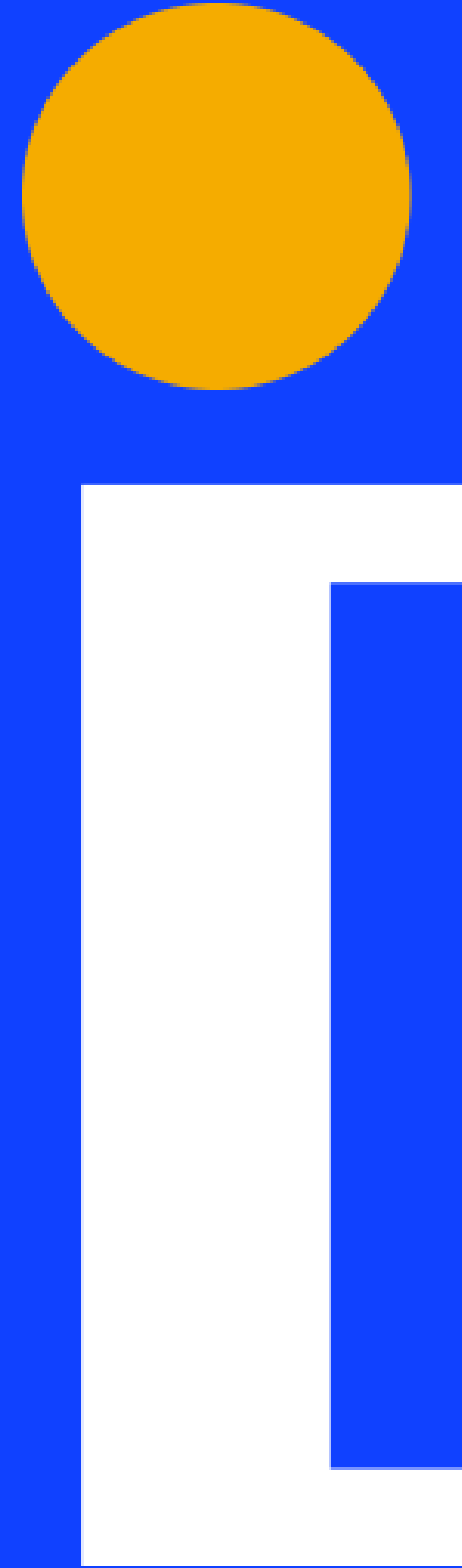
Disciplinas

- [Algoritmos e Programação de Computadores](#)
- [Lógica e Programação de Computadores](#)
- [Data Mining com Python](#)

Meus contatos

- E-mail: victor.silva@professores.ibmec.edu.br
- [Linkedin](#)
- [Lattes](#)

- O único arquivo exigido para o site é o **index.md**, que deve ficar na raiz do repositório. Novos arquivos e pastas podem ser criados se necessário, e a navegação é sempre relativa à raiz do repositório.



IBMEC.BR

 /IBMEC

 IBMEC

 @IBMEC_OFICIAL

 @IBMEC

 **ibmec**