pyinstaller

Live de Python # 228

Roteiro

1. Pyinstaller

Uma introdução. Histórico, instalação e funcionamento básico

2. Como isso funciona?

Como o pyinstaller empacota as coisas?

3. Runtime vs Runtime

Entendendo o bundle

4. Quando as coisas dão errado

Impor ocultos, hooks e bootloader

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriana Cavalcanti, Adriano Ferraz, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Allan Almeida, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, André Rafael, Aquiles Coutinho, Arnaldo Turque, Aurelio Costa, Bruno Batista, Bruno Divino, Bruno Freitas, Bruno Guizi, Bruno Lopes, Bruno Ramos, Caio Felix, Caio Nascimento, Carina Pereira, Christiano Morais, Clara Battesini, Dandara Sousa, Daniel Freitas, Daniel Haas, Daniel Santos, Danilo Segura, David Couto, David Kwast, Delton Porfiro, Denis Quirino, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Diogo Paschoal, Douglas Bastos, Douglas Zickuhr, Eduardo Tolmasquim, Emanuel Betcel, Emerson Rafael, Eneas Teles, Erick Ritir, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Everton Silva, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Fabricio, Fabricio Araujo, Felipe Rodrigues, Fernanda Prado, Fernando Florëncio, Firehouse, Flávio Meira, Flavkaze, Gabriel Barbosa, Gabriel Mizuno, Gabriel Nascimento, Gabriel Simonetto, Geandreson Costa, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Henrique Junqueira, Hugo Cosme, Igor Taconi, Ismael Ventura, Italo Silva, Izac Silva, Jairo Jesus, Jairo Lenfers, Janael Pinheiro, João Paulo, Joelson Sartori, Johnny Tardin, Jônatas Silva, José Barbosa, José Gomes, Joseíto Júnior, Jose Mazolini, José Pedro, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Júlio Pereira, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Lengo, Leonardo Mello, Leonardo Nazareth, Leon Solon, L. Perciliano, Luancomputacao Roger, Lucas Adorno, Lucas Carderelli, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Ratamero, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Luiz Paula, Maicon Pantoja, Maiquel Leonel, Marcelino Pinheiro, Márcio Martignoni, Marcio Moises, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Maria Gabriela, Marina Passos, Mateus Lisboa, Matheus Cortezi, Matheus Oliveira, Matheus Silva, Matheus Vian, Mauricio Fagundes, Mauricio Nunes, Mírian Batista, Mlevi Lsantos, Murilo Andrade, Murilocunha, Murilo Viana, Nando Sangenetto, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo D., Paulo Tadei, Pedro Henrique, Pedro Pereira, Pedro Silva, Peterson Santos, Priscila Santos, Rafael Lopes, Rafael Romão, Ramayana Menezes, Regis Santos, Regis Tomkiel, Rene Bastos, Ricardo Silva, Ricarte Jr, Riverfount, Rjribeiro, Robson, Robson Maciel, Rodrigo Alves, Rodrigo Cardoso, Rodrigo Freire, Rodrigo Messias, Rodrigo Quiles, Rodrigo Ribeiro, Rodrigo Vaccari, Rodrigo Vieira, Rogério Lima, Rogério Nogueira, Rogério Sousa, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Sebastião Tolentino, Stash, Talita Rossari, Tay Turnner, Thaynara Pinto, Thi, Thiago Araujo, Thiago Borges, Thiago Curvelo, Thiago Moraes, Thiago Souza, Tiago Minuzzi, Tiago Souza, Tony Dias, Tony Santos, Tyrone Damasceno, Uadson Emile, Valcilon Silva, Valdir Tegon, Vcwild, Vinicius Stein, Vitor Luz, Vladimir Lemos, Walter Reis, Wesley Mendes, Willian Lopes, Wilson Duarte, Wilson Neto, Wilson Rocha, Xico Silvério, Yury Barros

Obrigado você

Pyinstaller

Uma introdução

Pyinstaller

O PyInstaller agrupa um aplicativo Python e todas as suas dependências em um único pacote. O usuário pode executar o aplicativo empacotado sem instalar um interpretador Python ou qualquer módulo.

- História:
  - 2005 — Primeiro commit
  - 2015 — v1.0
  - 2022 — v5.7.0
- Licença: GPL2+ / Apache 2.0 (**Dual**)
- Suporta:
  - Linux, MacOS, Windows
  - AIX, Solaris, FreeBSD e OpenBSD

https://pyinstaller.org/en/v5.7.0/index.html

Requisitos

Embora o pyinstaller suporte todas esses sistemas operacionais. Alguns requisitos são necessários para a versão 5.7 (Isso varia de versão para versão)

- Windows 8+
- MacOS 10.15+ (Catalina+)
- Linux, ldd e glib C
- AIX, Solaris, FreeBSD e OpenBSD: ldd e objdump

Requisitos

Embora o pyinstaller suporte todas esses sistemas operacionais. Alguns requisitos são necessários para a versão 5.7 (Isso varia de versão para versão)

- Windows 8+
- MacOS 10.15+ (Catalina+)
- Linux, ldd e **glib C**
- AIX, Solaris, FreeBSD e OpenBSD: ldd e objdump

glibC em teoria é retro compatível. Se você deseja que o bundle funcione em sistemas mais antigos, garanta que está usando uma versão mais antiga da glibc

O que o pyinstaller não é?

1. Pyinstaller é um criador de "bundles" [agrupador] de pacotes. Isso quer dizer que ele **não é um compilador de código** Python como Cython, nuitka ou mypyc.

2. Pyinstaller **não é um otimizador de código**, um bundle criado com ele não é mais rápido que um código interpretado

3. Pyinstaller **não é um ofuscador de código**. É possível fazer engenharia reversa no bundle gerado.

4. Pyinstaller **não é um criador de instaladores**. Você deve conduzir isso a parte com ferramentas como inno setup, NSIS, etc. para instalar seu bundle

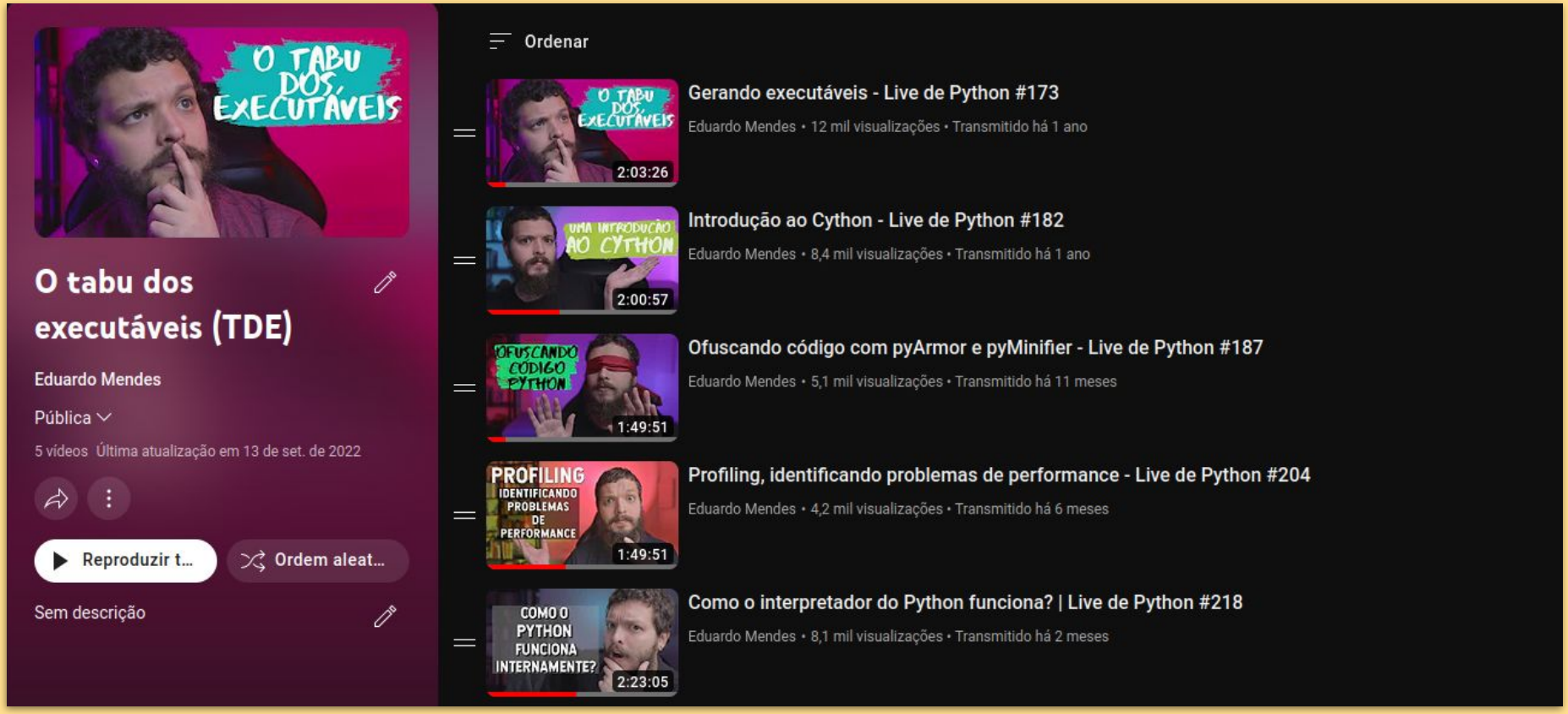
Plataformas

Por varrer as dependências da sua instalação e usar o seu python como base. O executável gerado **não é multiplataforma** (cross-system). Isso quer dizer que um executável gerado no Linux, só funcionará no Linux. No Windows somente no Windows.

Tanto as bibliotecas e o interpretador são compilados para arquiteturas diferentes. Como 32-bits e 64-bits. Então um bundle gerado em um Linux 64bits não funcionará em um Linux 32bits. O mesmo vale para os outros sistemas.

Como bibliotecas (.so e DDLs) são coletadas do sistema. Em teoria, versões mais antigas de sistemas são propensas a maior compatibilidade entre sistemas mais novos. Ex: Bundle de Win8 funciona no Win11. Porém um bundle criada em Win11, não funciona no Win8 em alguns casos

Esse vídeo faz parte de uma série



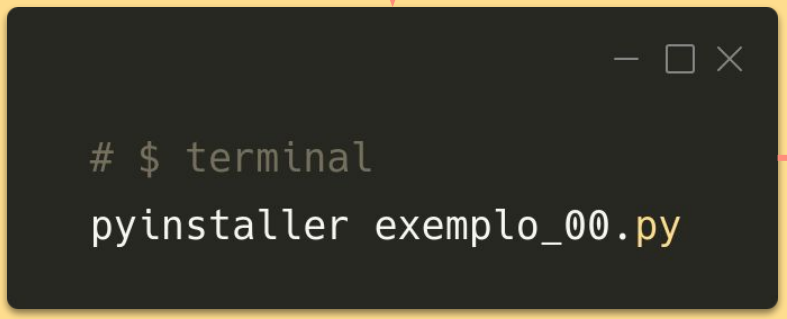
https://www.youtube.com/playlist?list=PLOQgLBuj2-3Kpx-Qaf00mVUUmSgv4Vinx

pip install pyinstaller

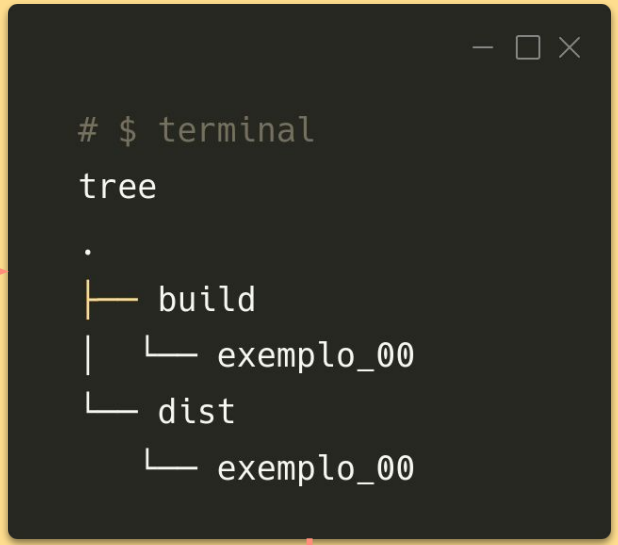
Instalação

Ciclo básico de funcionamento

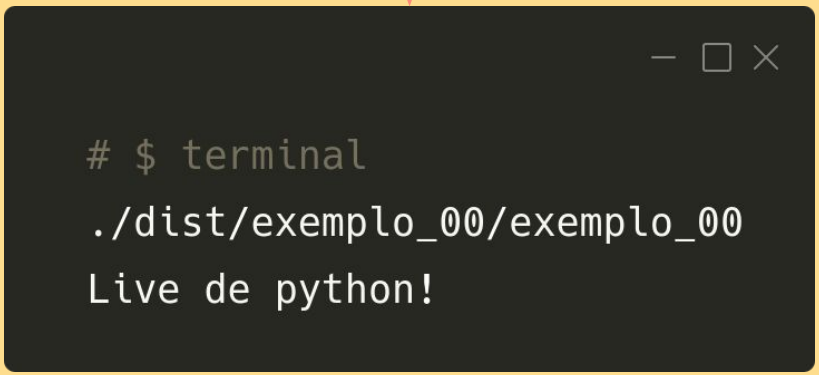
```python
# exemplo_00.py
print('Live de Python!')
```



```
# $ terminal
pyinstaller exemplo_00.py
```



```
# $ terminal
tree
.
├── build
│   └── exemplo_00
└── dist
    └── exemplo_00
```



```
# $ terminal
./dist/exemplo_00/exemplo_00
Live de python!
```

Ciclo básico de funcionamento

```python
# exemplo_00.py
print('Live de Python!')
```

build é uma pasta de trabalho (workpath). São arquivos temporários do build. Podem ser deletados

```
# $ terminal
tree
.
├── build
│   └── exemplo_00
└── dist
    └── exemplo_00
```

```
# $ terminal
./dist/exemplo_00/exemplo_00
Live de python!
```

Ciclo básico de funcionamento

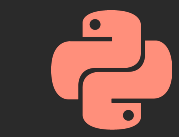
```python
# exemplo_00.py
print('Live de Python!')
```

```
# $ terminal
tree
.
├── build
│   └── exemplo_00
└── dist
    └── exemplo_00
```

Dist é o diretório de "distribuição". Dentro dele são criadas as pastas com os bundles

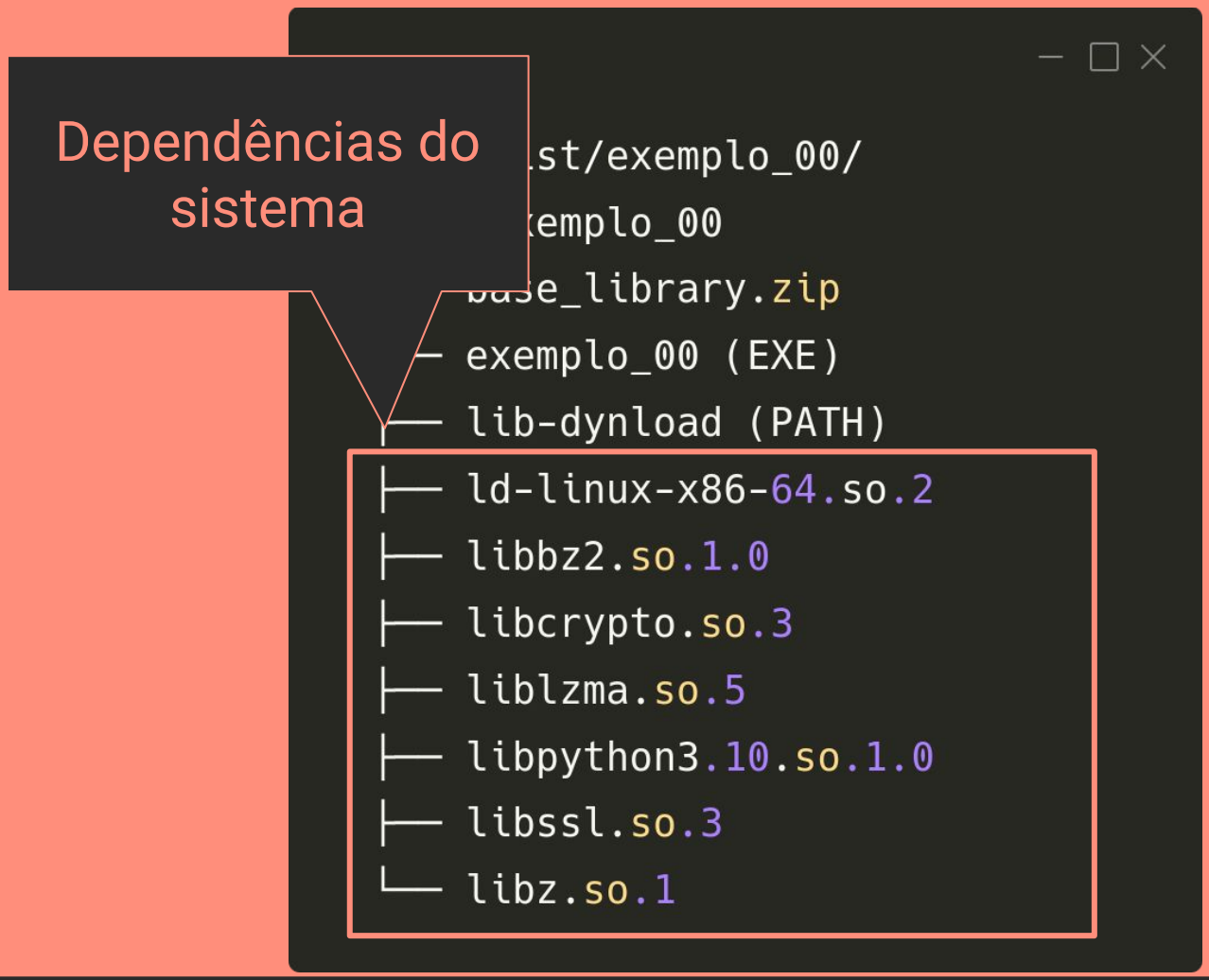
```
# $ terminal
./dist/exemplo_00/exemplo_00
Live de python!
```

O que vem no bundle?

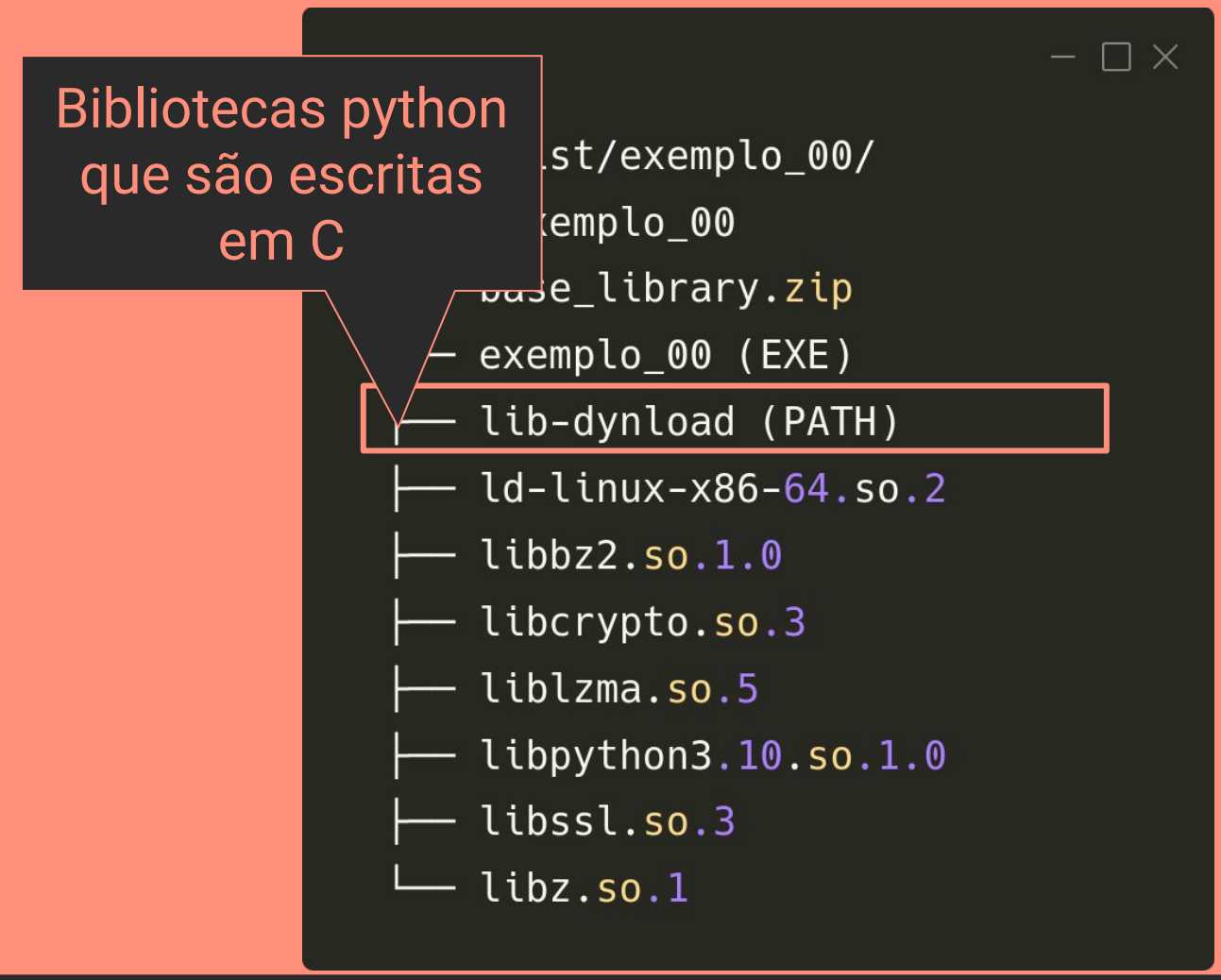


```
tree dist/exemplo_00/
dist/exemplo_00
├── base_library.zip
├── exemplo_00 (EXE)
├── lib-dynload (PATH)
├── ld-linux-x86-64.so.2
├── libbz2.so.1.0
├── libcrypto.so.3
├── liblzma.so.5
├── libpython3.10.so.1.0
├── libssl.so.3
└── libz.so.1
```

O que vem no bundle?



O que vem no bundle?



O que vem no bundle?

```
tree dist/exemplo_00/
dist/exemplo_00
├── base_library.zip
├── exemplo_00 (EXE)
├── lib-dynload (PATH)
│   ├── linux-x86-64.so.2
│   ├── bbz2.so.1.0
│   ├── bcrypto.so.3
├── liblzma.so.5
├── libpython3.10.so.1.0
├── libssl.so.3
└── libz.so.1
```

Nosso executável

O que vem no bundle?

Bibliotecas python em python (.py)

Formas de gerar um bundle

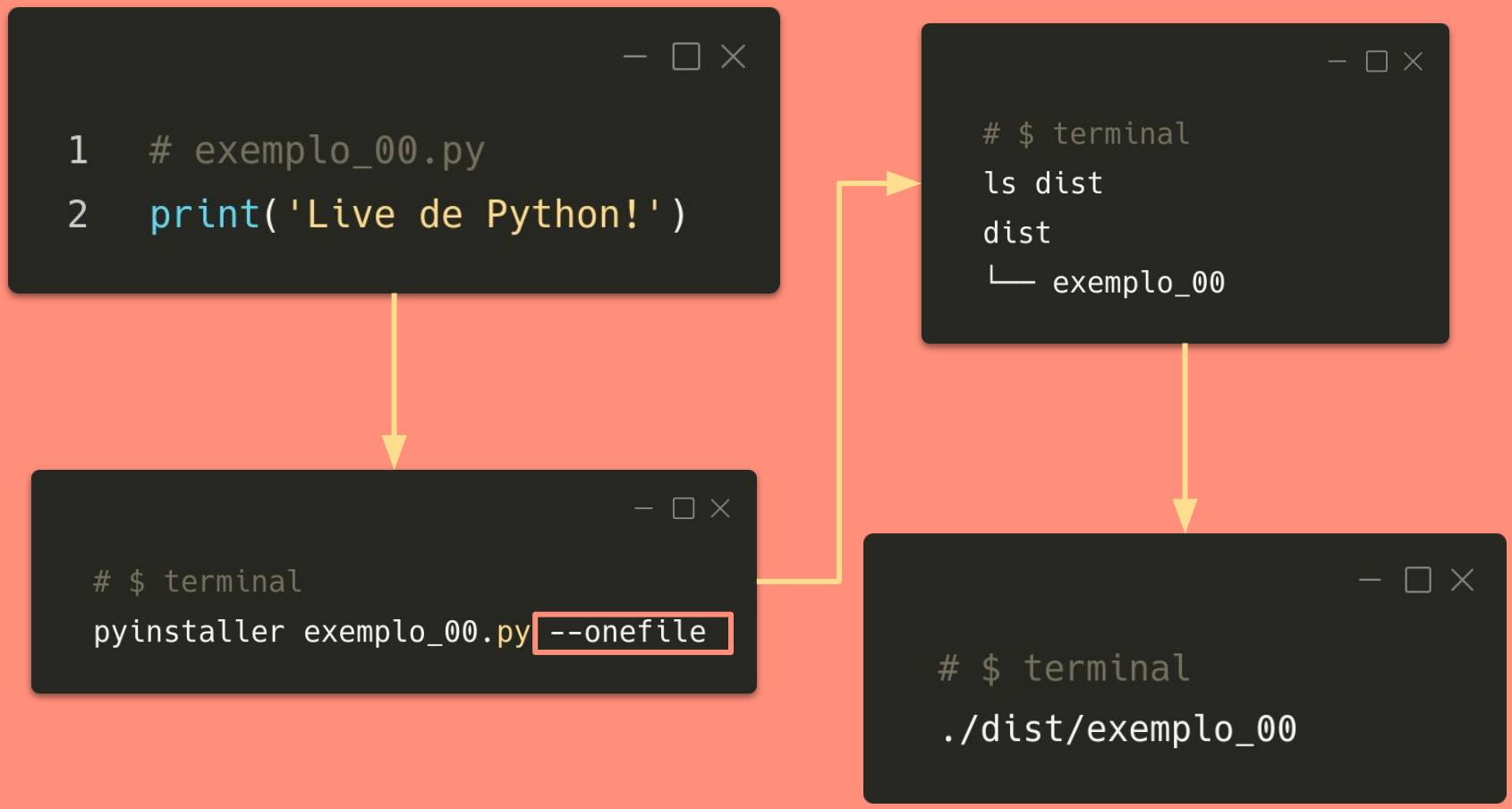
O pyinstaller básicamente permite duas formas de criação de bundles:

- **Um diretório**: Arquivo executável acompanhado de suas dependências
- **Portável**/Arquivo único: Um único executável

Essas diferenças são significativas para o modo em que você vai distribuir o seu bundle.

Um diretório pode ser compactado (um zip, por exemplo) e distribuído e o arquivo portável já é um executável pronto para distribuição.

Gerando um portável



Os dois lados da moeda (diretórios)

Embora os dois bundles funcionem eficientemente, existem algumas vantagens e desvantagens.

Vantagens do diretório:

- O processo de inicialização é mais rápido
- O build leva menos tempo, caso as bibliotecas instaladas não mudem
- Mais fácil de debugar

Desvantagens do diretório:

- Precisa ser compactado para ser distribuído
- Podem deixar pessoas quem usam confusas com a abundância de arquivos

Os dois lados da moeda (portáveis)

Embora os dois bundles funcionem eficientemente, existem algumas vantagens e desvantagens.

Vantagens do portável:

- Um único arquivo precisa ser distribuído

Desvantagens do portável:

- Desempacotamento
- Executáveis maiores
- Facilidade em ser detectado por antivírus

O problema de desempacotar

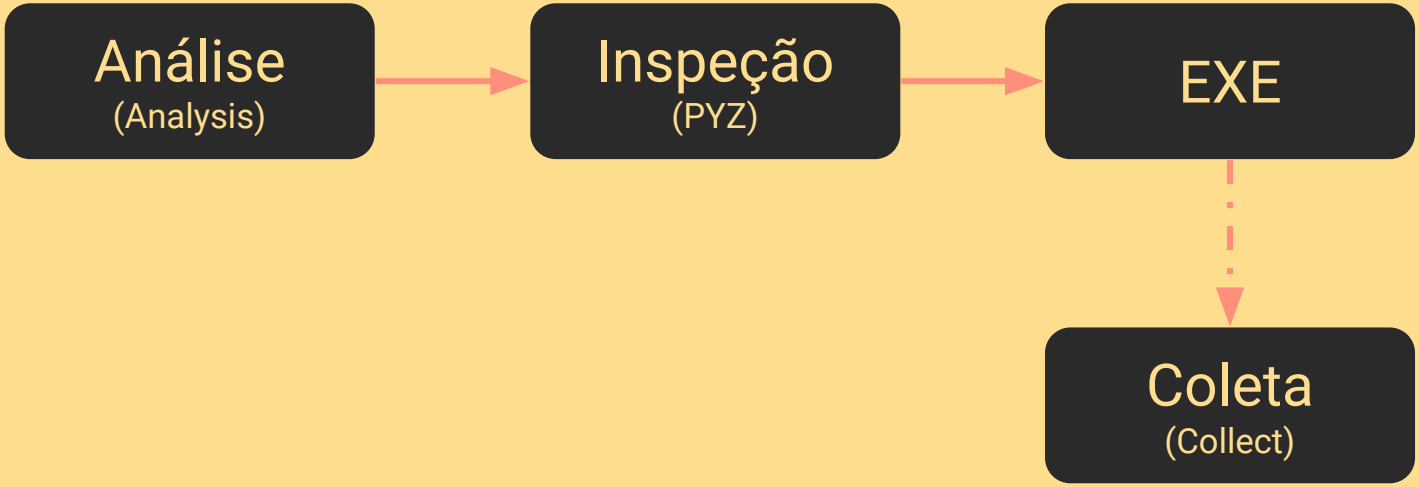
- As bibliotecas vão ser desempacotadas do executável todas às vezes na pasta temporária
- As bibliotecas serão deletadas quando o executável for fechado
- Quando abrir o app, sempre será lento
- Em pcs mais fracos, deletar os temporários pode levar a um consumo elevado de disco

O desempacotamento acontece em um diretório chamada **_MEIxxxxxx**, onde xxxxxx é um valor randômico.

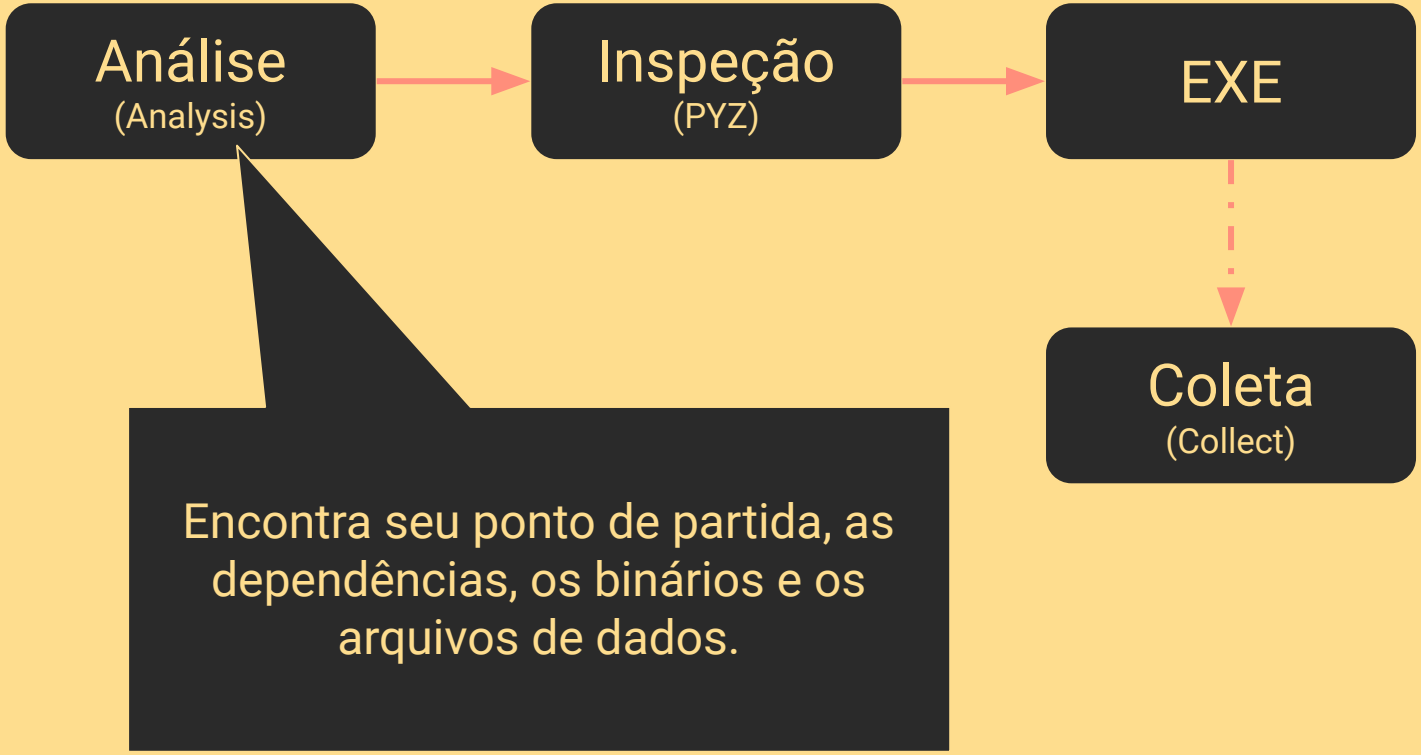
Como funciona?

Como o pyinstaller empacota?

O empacotamento é feito em 3 ou 4 partes



O empacotamento é feito em 3 ou 4 partes

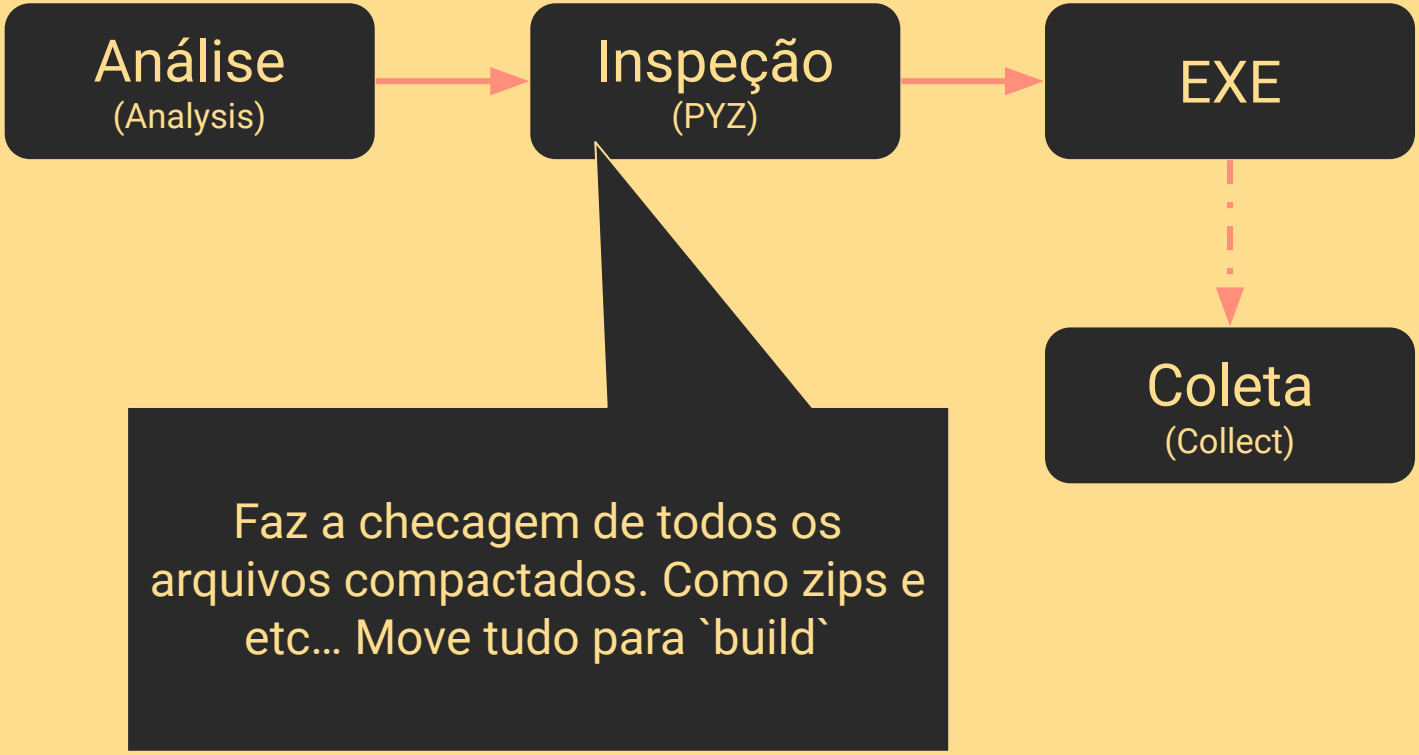


A fase de análise

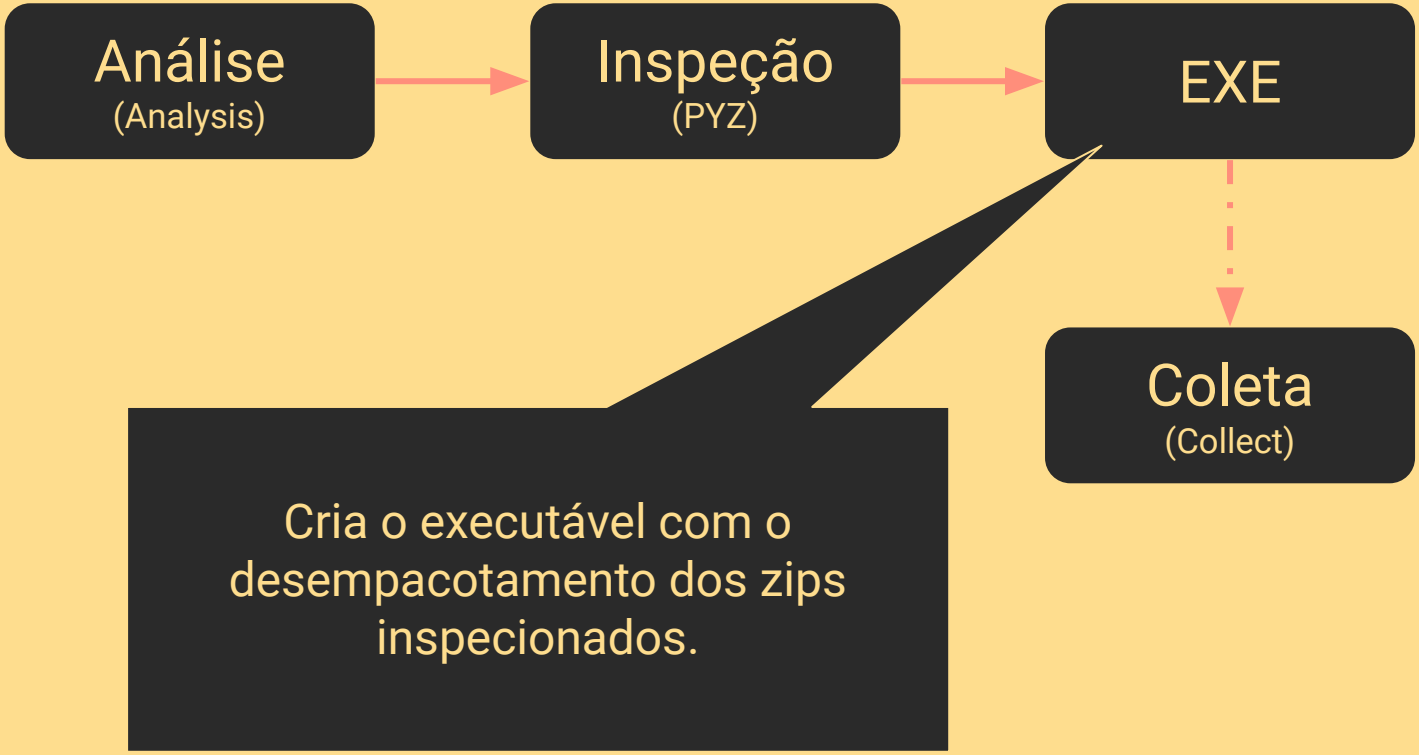
Inicialmente o pyinstaller recebe um arquivo com parâmetro. E dessa forma vai **percorrendo todos os imports*** e buscar no sitepackages (ver live 191 Ambientes virtuais e instalação de bibliotecas). Dessa forma criando um grafo de dependências.

Com as dependências prontas, ele busca o interpretador, copia todos os arquivos para o diretório `build` e os condensa em um arquivo executável com o nome do arquivo que recebeu como parâmetro.

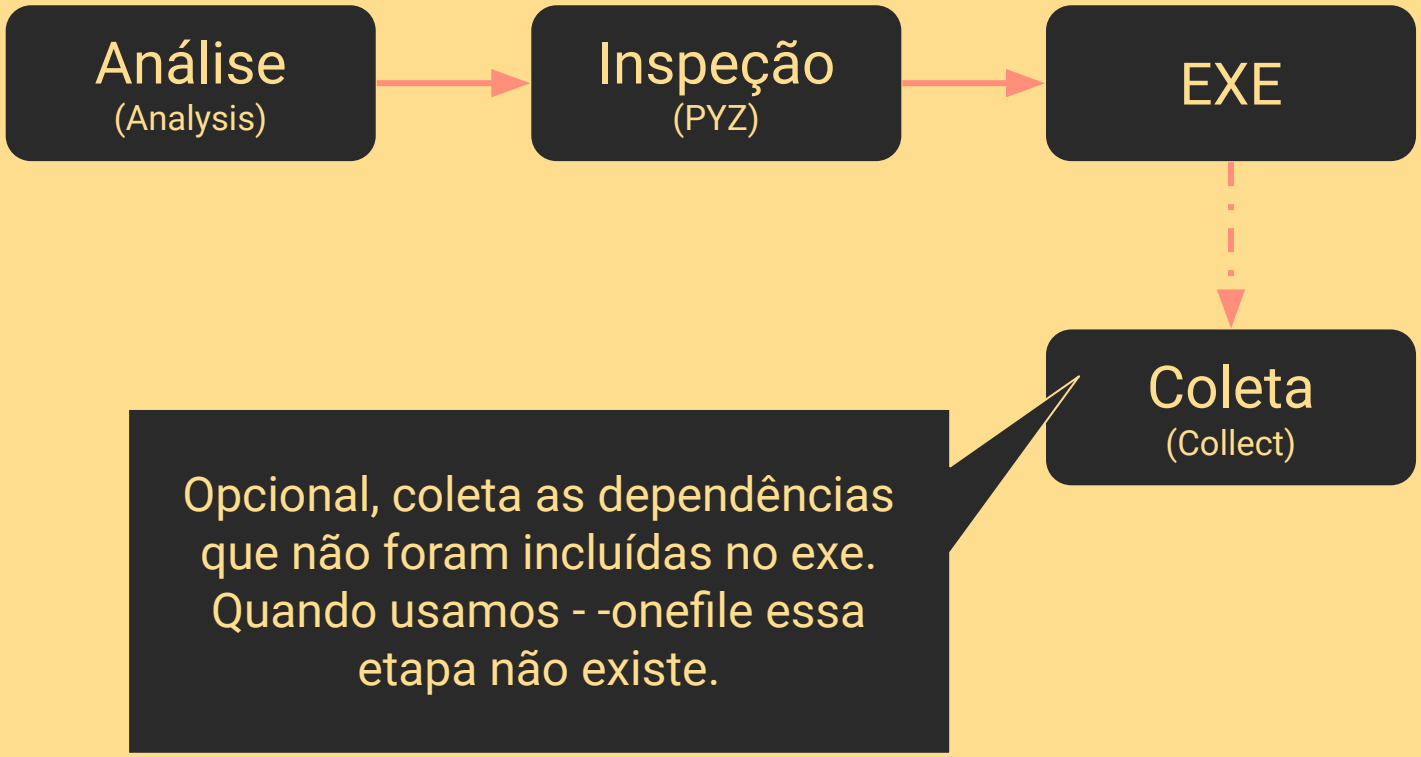
O empacotamento é feito em 3 ou 4 partes



O empacotamento é feito em 3 ou 4 partes



O empacotamento é feito em 3 ou 4 partes



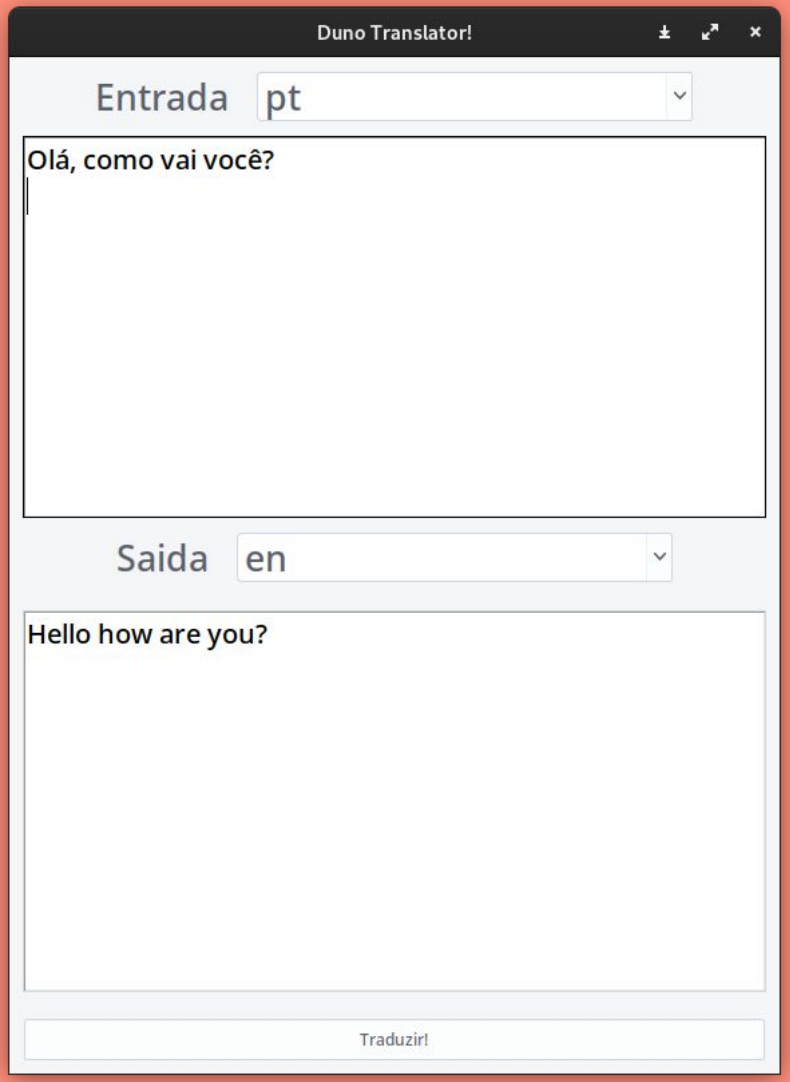
Arquivo .spec

Quando geramos o nosso primeiro bundle, o aquivo spec é criado. Nele podemos ver de forma descritiva o que é usado em cada etapa. Caso queira criar um spec sem o bundle, pode usar o CLI do pyinstaller

```
1   # Terminal
2   pyi-makespec script
```

Entendendo o bundle

Runtime vs runtime



https://youtu.be/vNEwbfsZ-Js

Path base

O diretório de referência muda. Por exemplo. Quando executamos o python, o diretório raiz da execução é onde o interpretador foi chamado.

Porém, no executável esse caminho pode ser qualquer lugar onde estiver sendo executado. Por conta disso, o pyinstaller injeta uma variável chamada MEIPASS no sys. Para termos uma referência de partida dos arquivos.

sys._MEIPASS

Runtime do Python vs Runtime do pyinstaller

Precisamos de uma forma do código saber se está sendo executado pelo interpretador ou está rodando no bundle. Para isso, o pyinstaller também injeta uma variável chamada `frozen` no sys. Um snippet que pode te ajudar com isso:

```python
import sys
from pathlib import Path

def base_path() -> Path:
    if getattr(sys, 'frozen', False) and hasattr(sys, '_MEIPASS'):
        return Path(sys._MEIPASS).resolve()
    return Path('.').resolve()
```

Quando as coisas dão errado

Problemas

Hidden imports

Em alguns momentos podemos encontrar com códigos que fazem importação dinâmica. Como sys.path, __import__ ou importlib.

Para esses contextos o pyinstaller perde a referência e não consegue fazer os imports corretamente.

```
a = Analysis(
    ['tradutor_03.py'],
    pathex=[],
    binaries=[],
    datas=[('langs.json', '.')],
    hiddenimports=['tools'],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)
```

Bootloader

Bootloader é um binário capaz de iniciar a máquina vitual do python modificada pelo pyinstaller. Em alguns casos é necessário compilar um novo bootloader:

- Seu sistema não é oficialmente suportado (BSDs, AIX, versões novas)
- O bootloader disponível não funciona em determina versão X de aplicativos do seu sistema
- Windows reconhece como vírus

https://pyinstaller.org/en/latest/bootloader-building.html

Hooks

Algumas bibliotecas têm comportamentos bastante inusitados. Às vezes fazem compilação dentro delas, não dependem só de python e são bastante complicadas de empacotar. Por exemplo, pandas e SQLAlchemy.

Para isso, a comunidade do pyinstaller provêm diversos hooks para empacotar essas bibliotecas

- Nativos: https://github.com/pyinstaller/pyinstaller/tree/develop/PyInstaller/hooks
- Comunidade: https://github.com/pyinstaller/pyinstaller-hooks-contrib
- Como criar hooks: https://github.com/pyinstaller/hooksample
- Pacotes suportados: https://github.com/pyinstaller/pyinstaller/wiki/Supported-Packages

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3