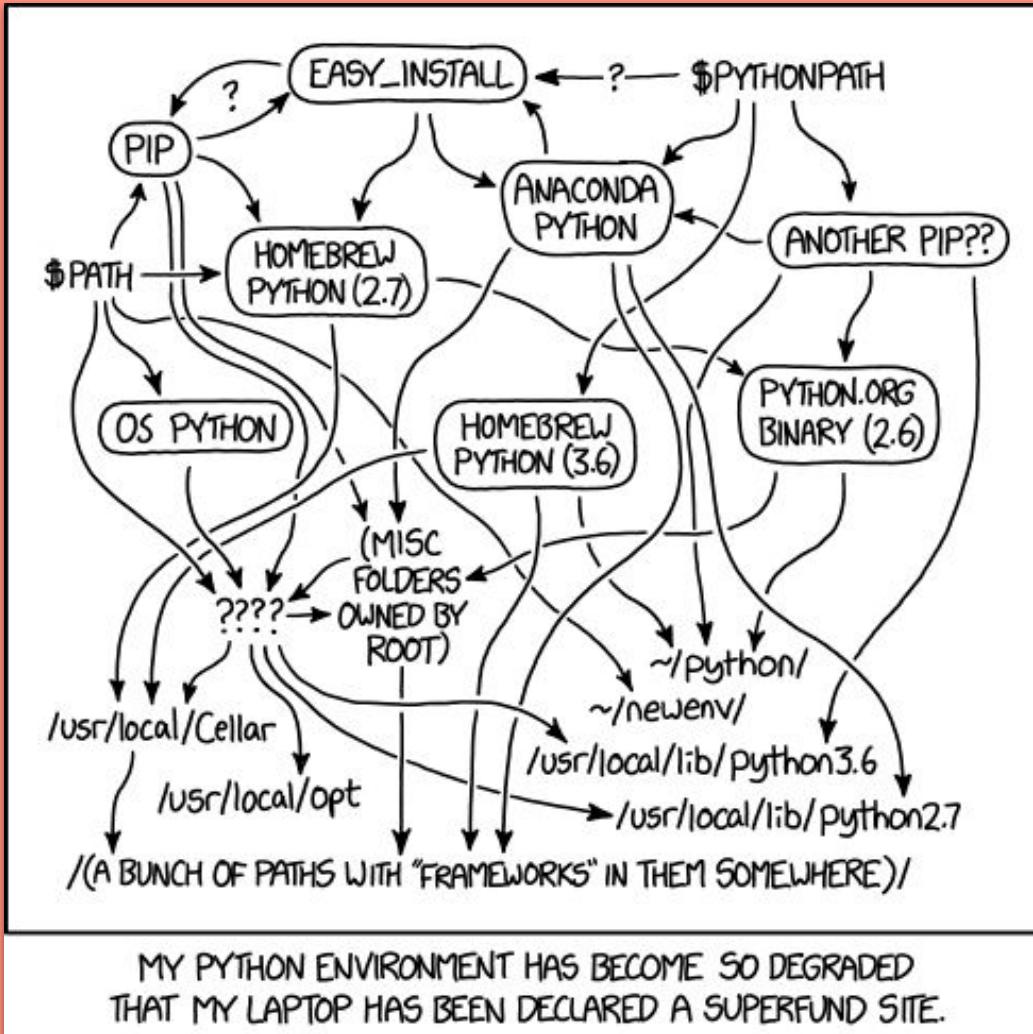


pyproject.toml

Live de Python #257



<https://xkcd.com/1987/>

Eu legitimamente acho Python uma ótima linguagem. E eu acho que nos últimos anos o ecossistema tem se tornado cada vez mais interessante. E mesmo com tudo isso, eu deixaria de usar Python só pra não ter que lidar com o pip.

15:48 · 21 jan. 24 · 31,1K Visualizações

<https://encurtador.com.br/lyD79>

Live sobre build-systems #404

 Closed henriquebastiao opened this issue on Dec 25, 2023 · 0 comments



henriquebastiao commented on Dec 25, 2023

Seria interessante uma live sobre build-systems, uma discussão que foi iniciada no grupo do Telegram. Como usar pip com arquivos `pyproject.toml`, etc...



Sugestão de tema para live - utilizando `.toml` no projeto #403

 Open suportebeloj opened this issue on Dec 25, 2023 · 1 comment



suportebeloj commented on Dec 25, 2023

A ideia seria abordar o `.toml` como alternativa ao `requirements.txt`, viabilizando o gerenciamento de pacotes de maneira inteligente, como também informações do projeto estilo o que o node tem com o `package.json`.

Vale ressaltar a importância de não usar terceiros como `poetry` por exemplo, a ideia aqui é gerenciar esses pacotes de maneira nativa ou o mais próximo disso, e agilizar o deploy de aplicações com base nos ambientes definidos pelo dev.

A ideia surgiu em uma discussão no grupo do Telegram



Poetry é melhor

PDM, pfv

Eu uso pip

hatch!

rye!



QUEM SOMOS?



PYTHONISTAS



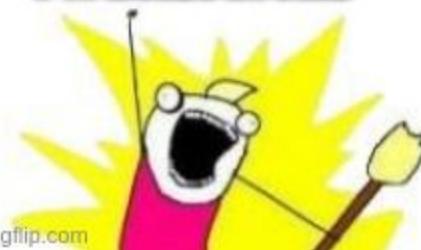
O QUE QUEREMOS?



**UMA FERRAMENTA
ÚNICA PARA PACOTES**



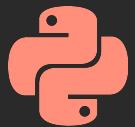
PYPROJECT.TOML



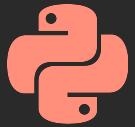
imgflip.com

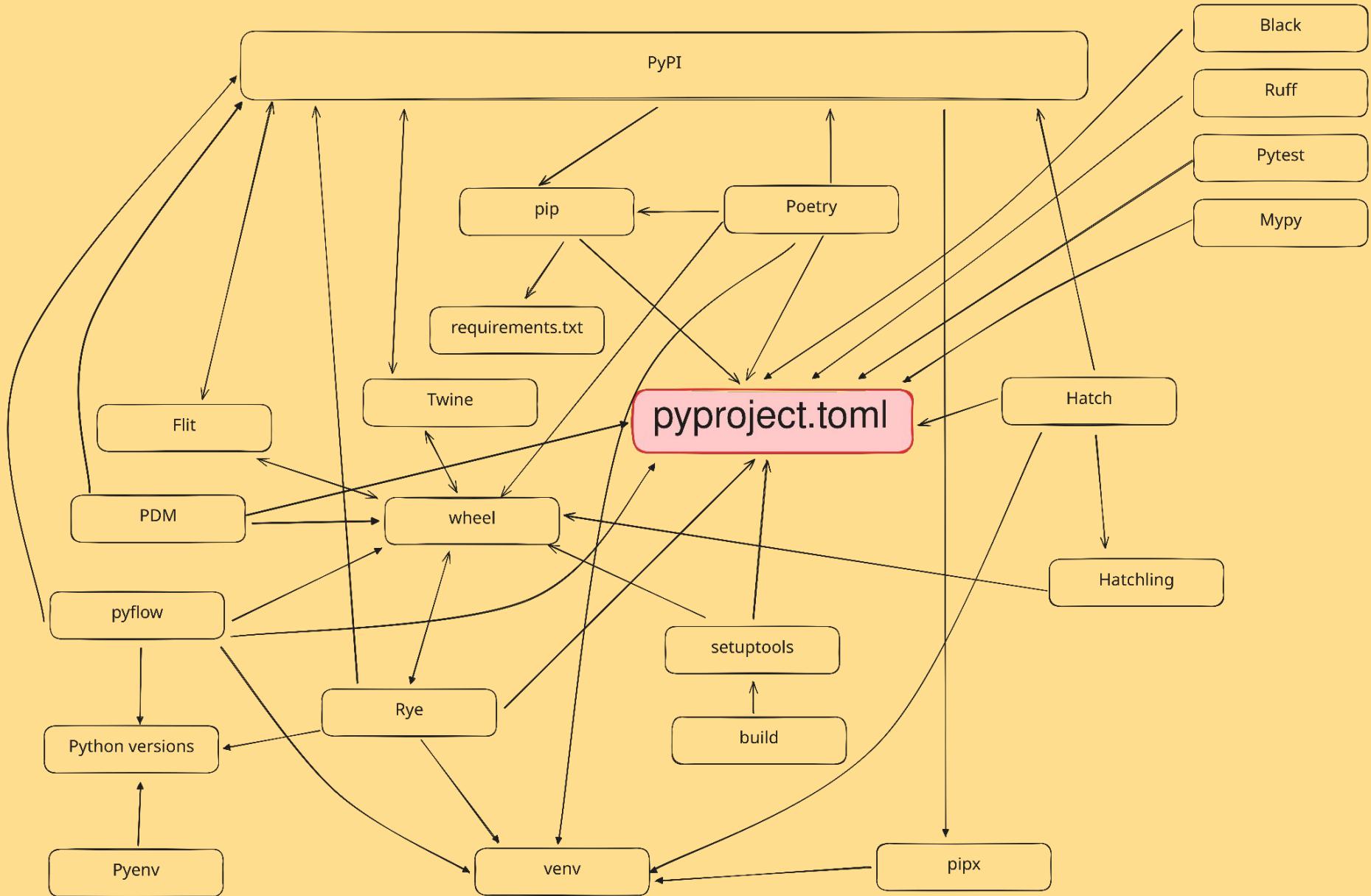
<https://imgflip.com/i/8fl920>

USE O pyproject.toml



Minha resposta padrão para tudo





Q?



apoia.se/livedepython



pix.dunossauro@gmail.com



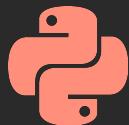
patreon.com/dunossauro



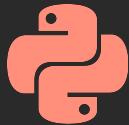
Ajude o projeto <3



Ademar Peixoto, Adilson Herculano, Alemao, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alfredo Braga, Alisson Souza, Alysson Oliveira, Andre Azevedo, Andre Mesquita, Andre Paula, Aniltair Filho, Antônio Filho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bárbara Grillo, Ber_dev_2099, Bernardo Fontes, Bruno Almeida, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Ramos, Caio Nascimento, Carlos Ramos, Cristian Firmino, Daniel Bianchi, Daniel Freitas, Daniel Real, Daniel Wojcickoski, Danilo Boas, Danilo Silva, David Couto, David Kwast, Davi Souza, Dead Milkman, Denis Bernardo, Diego Guimarães, Dino, Edgar, Edilson Ribeiro, Emerson Rafael, Ennio Ferreira, Erick Andrade, Érico Andrei, Everton Silva, Fabio Barros, Fábio Barros, Fabio Valente, Fabricio Biazzotto, Felipe Augusto, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Flávio Meira, Francisco Silvério, Frederico Damian, Gabriel Espindola, Gabriel Mizuno, Gabriel Moreira, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Giovanna Teodoro, Giuliano Silva, Guilherme Beira, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Gustavo Suto, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Henrique Andrade, Hiago Couto, Higor Monteiro, Igor Taconi, Janael Pinheiro, Jean Victor, Jefferson Antunes, Joelson Sartori, Jonas Leon, Jônatas Oliveira, Jônatas Silva, Jorge Silva, Jose Barroso, José Gomes, Joséito Júnior, Jose Mazolini, Josir Gomes, Juan Felipe, Juliana Machado, Julio Batista-silva, Julio Franco, Júlio Gazeta, Júlio Sanchez, Kaio Peixoto, Leandro Silva, Leandro Vieira, Leonardo Mello, Leonardo Nazareth, Lucas Carderelli, Lucas Lattari, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Eduardo, Luiz Carlos, Luiz Duarte, Luiz Lima, Luiz Paula, Mackilem Laan, Marcelo Araujo, Marcio Moises, Marcio Silva, Marco Mello, Marcos Gomes, Marina Passos, Mateus Lisboa, Matheus Ferreira, Matheus Silva, Matheus Vian, Mírian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Pedro Gomes, Pedro Henrique, Pedro Pereira, Peterson Santos, Rafael Araújo, Rafael Faccio, Rafael Lopes, Rafael Romão, Rafael Silva, Raimundo Ramos, Ramayana Menezes, Renato José, Renato Moraes, Renato Oliveira, Renê Barbosa, Rene Pessoto, Renne Rocha, Ricardo Combat, Ricardo Silva, Rinaldo Magalhaes, Riverfount, Rjribeiro, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Santana, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rui Jr, Samanta Cicilia, Samuel Santos, Santhiago Cristiano, Selmison Miranda, Téo Calvo, Thiago Araujo, Thiago Borges, Thiago Curvelo, Tony Dias, Tyrone Damasceno, Valdir, Valdir Tegon, Vinícius Costa, Vinicius Stein, Washington Teixeira, Willian Lopes, Wilson Duarte, Zeca Figueiredo



Obrigado você



Roteiro



1. Entendendo o problema

A complexidade do ecossistema do python

2. Uma breve, mas longa história

O legado das aplicações que vieram antes e ainda são mantidos

3. `pyproject.toml`

PEP-517, PEP-518, PEP-621, PEP-658

4. O que ainda deve ser feito?

Nem todos os problemas estão resolvidos

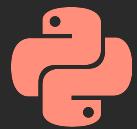
Proble mas!!!

Tudo que envolve
empacotamento

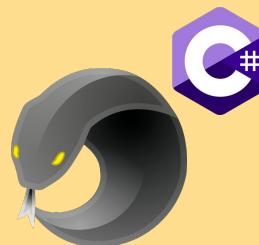
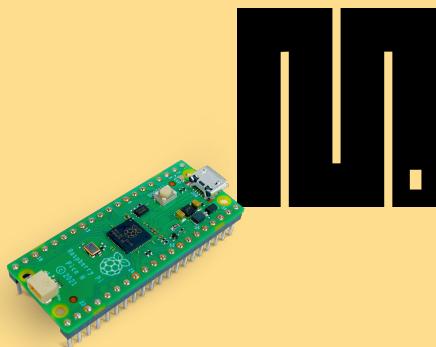
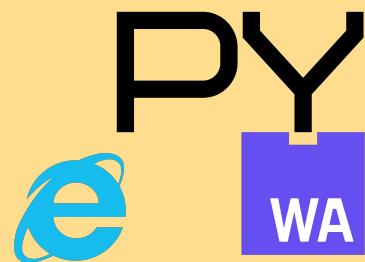
Python tem um ecossistema complexo



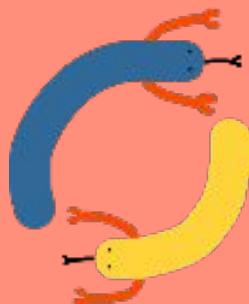
Python tem um ecossistema complexo



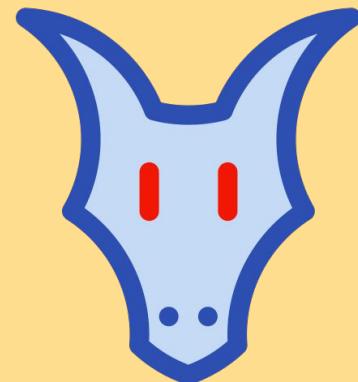
Python roda em lugares muito diferentes



Python tem diferentes implementações



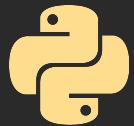
Python está contido em muitos lugares



Cont exto

Um pouco de
história!

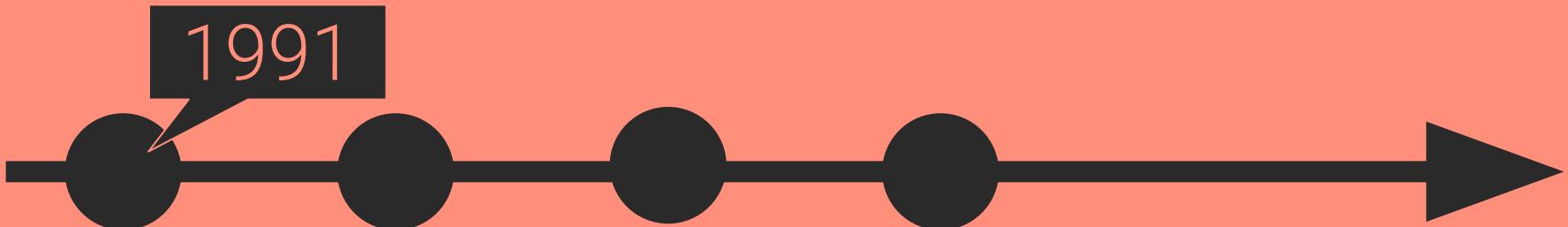
Eu sou um pythonista relativamente novo, muitas das coisas que eu vou falar aqui são por que eu li ou ouvi pessoas mais antigas na linguagem comentado!



Um ponto!



- Python foi criado em 1991, numa situação **muito** diferente de linguagens "modernas".
- A distribuição de pacotes era completamente diferente!
- Python não contava com uma forma padrão de distribuição de pacotes. *Até por que, vai por onde?*

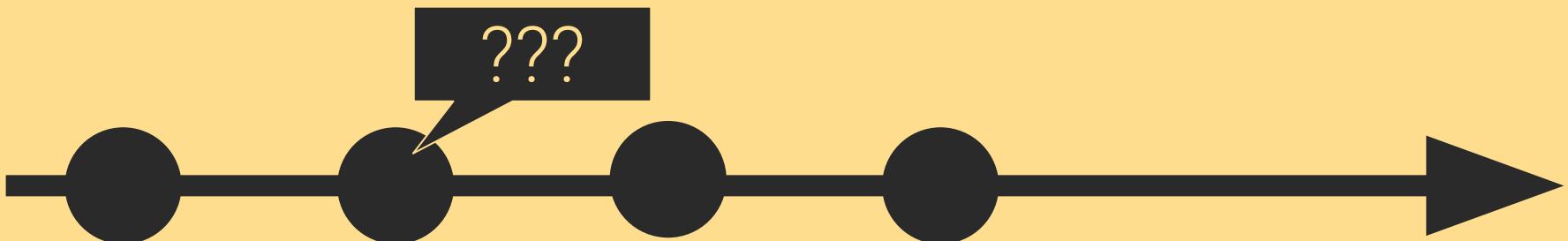


- Um lugar para indexar pacotes foi criado, Vaults of Parnassus

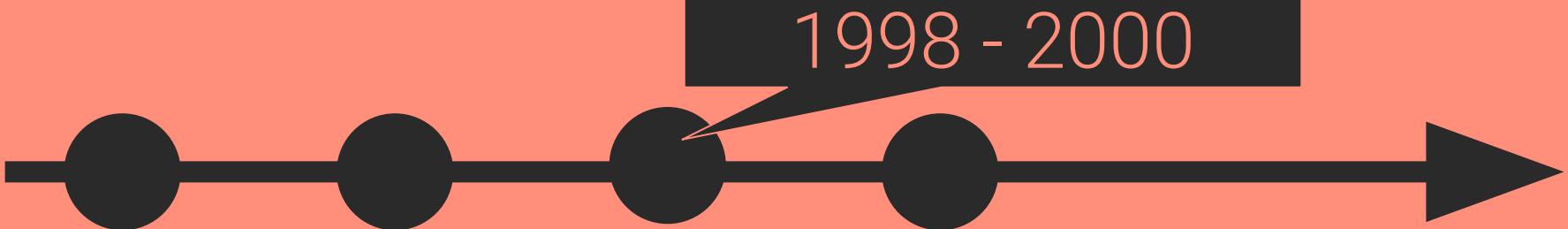
The screenshot shows the homepage of the 'Vaults of Parnassus' website. The title 'Vaults of Parnassus : Python Resources.' is at the top, followed by a navigation bar with links: About, Feedback, Latest, Random, Stats, Submit, and Tree. On the left, there's a small image of a book and a search bar labeled 'Find:' with a magnifying glass icon. The main content area is titled 'Vault:' and contains a list of categories with their counts:

➤ Applications - Ready to run programs written in, or using, Python.	(5 86)
➤ Database - Database related.	(95)
➤ Games - And game systems.	(61)
➤ Graphics - Graphics processing, manipulation, display, etc.	(1 95)
➤ Info/Books/Tutorials - In a word: reading material.	(3 1)
➤ Internet - Specifically Internet related networking stuff.	(6 38)
➤ Lost/Broken Links - Broken Links and things that need updating. Can you help?	(26)
➤ Math - Mmmmm, crunchy.	(1 94)
➤ Miscellany - Odds and ends, and otherwise uncategorised things.	(101)
➤ Networking - Networks, distributed computing, sockets, etc.	(3 39)
➤ O/S Support - Modules and tools specifically targetting various operating systems.	(1 47)
➤ Programming Tasks - Datatypes, parsing, algorithms, files, etc.	(4 18)
➤ Python Tools/Extensions - Patches and other enhancements for making Python behave the way you want.	(2 74)
➤ Sound/Audio - More than a slight rustling in the grass.	(64)
➤ User Interfaces - Text and GUI interfaces, widgets and controls for use with python.	(2 9)
➤ Web Sites - Python related web sites (personal, user group, non-english, collections, etc).	(3 30)

Parnassus Totals: 2025 items in 49 categories.



- Foi criado um SIG (Special Interest Groups) para distribuição de pacotes
- Disso surgiu o **Distutils**
- Ferramenta embutida até a versão 3.11
- Introdução do arquivo **setup.py**
- setup.py era responsável pelo **build** e pela **instalação** do pacote
- Os formatos de código (**sdist**) e binário (**bdist**) foram convencionados



1998 - 2000

A timeline diagram at the bottom of the slide features a horizontal black arrow pointing to the right, representing time. Four dark gray circular markers are placed along the arrow. A speech bubble-shaped callout box is positioned above the third marker from the left. The text "1998 - 2000" is written in white inside this callout box. A thin black line extends from the top edge of the callout box to the third circular marker, indicating the specific time period being discussed.

Distutils



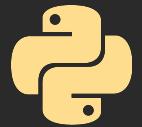
```
1 # setup.py
2 from distutils.core import setup
3
4 setup(
5     name='package',
6     version='0.0.1',
7     # O pacote deve ter um `__init__.py`
8     # para ser reconhecido como um módulo.
9     packages=['app'],
10 )
```

Distutils



```
1 # setup.py
2 from distutils.core import setup
3
4 setup(
5     name='package',
6     version='0.0.1',
7     # O pacote deve ter um `__init__.py`
8     # para ser reconhecido como um módulo.
9     packages=['app'],
10 )
```

metadados!



Distutils

```
1 # setup.py
2 from distutils.core import setup
3
4 setup(
5     name='package',
6     version='0.0.1',
7     # O pacote deve ter um `__init__.py`
8     # para ser reconhecido como um módulo.
9     packages=['app'],
10 )
```

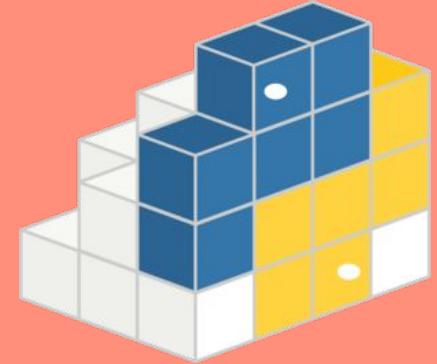
```
# para distribuição do fonte
python setup.py sdist
```

```
# para distribuição binária
python setup.py bdist
```

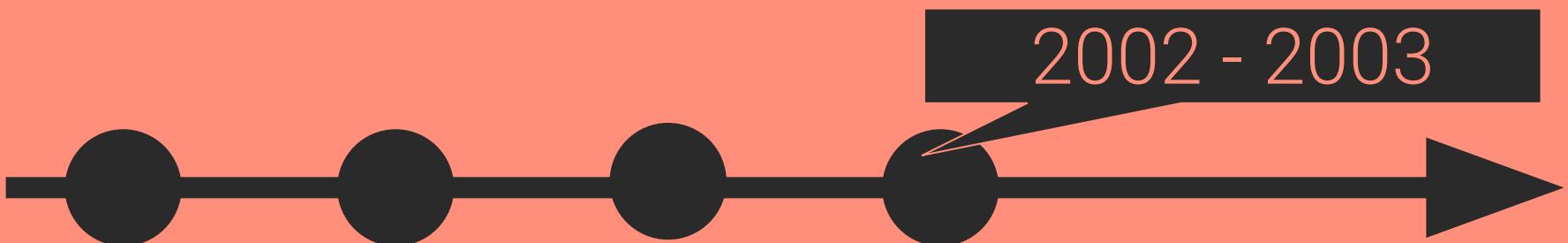
Distutils



```
.  
├── app  
│   ├── __init__.py  
│   └── __main__.py  
├── dist  
│   ├── package-0.0.1.linux-x86_64.tar.gz  # bdist  
│   └── package-0.0.1.tar.gz                 # sdist  
└── setup.py
```



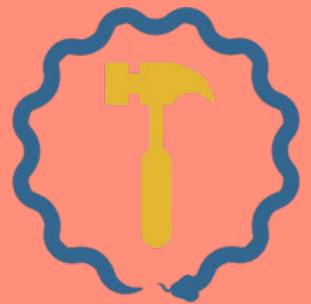
- Início do **Cheese Shop** (2002)
- Publicação do Python Package Index [**PyPI**] (2003)
- Ainda era só um **index**
- Os pacotes só começaram a ser armazenados em 2005
- **PEP 301** conta sobre os padrões



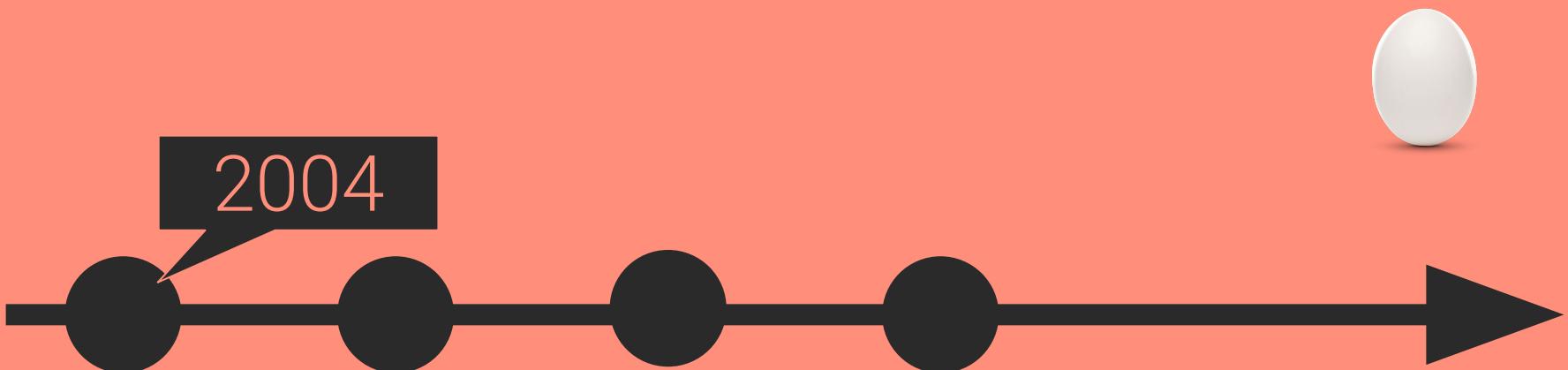


Do you have any cheese?

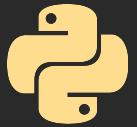
Cheese Shop sketch (1972) - Monty Python



- Introdução do **setuptools**
- Estende as funcionalidades do distutils
- Declaração de bibliotecas extensas no setup.py
- **easy_install** foi apresentado como uma ferramenta simples de instalação de pacotes (*só instalar, não dava pra desinstalar ???*)
- Introdução do formato **eggs** para binários
 - Eggs são para python como jars estão para java
- *setuptools ainda é a ferramenta "padrão" (ela se adaptou a todas as mudanças)*



SetupTools

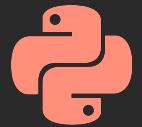


- □ ×

```
# setup.py
from setuptools import setup

setup(
    name='package',
    version='0.0.1',
    packages=[ 'app' ],
    install_requires=['rich', 'rich-pixels', 'pillow']
)
```

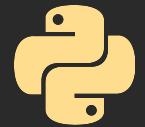
Dependencias!



Setup tools

```
.  
├── app  
│   ├── __init__.py  
│   └── __main__.py  
├── dist  
│   ├── package-0.0.1-py3.9.egg  
│   ├── package-0.0.1.linux-x86_64.tar.gz  
│   └── package-0.0.1.tar.gz  
└── setup.py
```

```
python setup.py bdist_egg  
python setup.py bdist  
python setup.py sdist
```

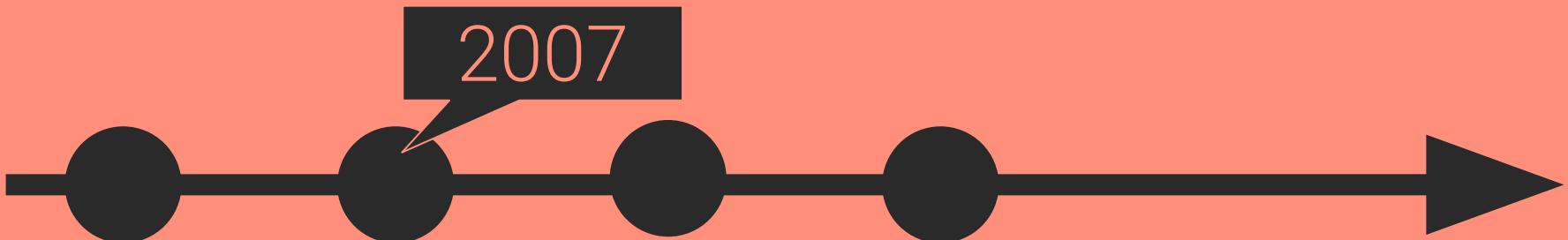


easy_install

- □ ×

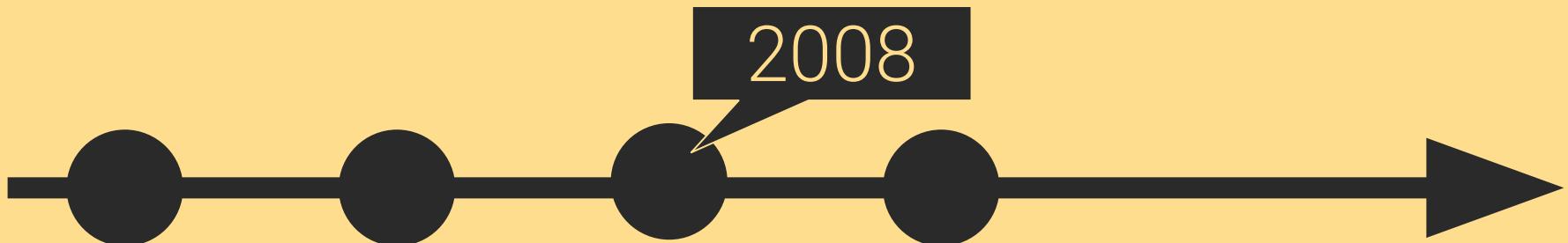
```
# procurando os pacotes no index do PyPI
easy_install pacote
# faz o download e o build do sdist
easy_install http://example.com/path/to/MyPackage-1.2.3.tgz
# instala um .egg local
easy_install /my_downloads/OtherPackage-3.2.1-py2.3.egg
# instala o pacote local com setuptools
# (baixa as dependências e faz o build de cada uma)
easy_install .
```

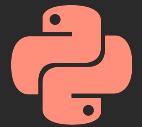
- Introdução do **virtualenv**
- virtualenv vs **-m venv**
 - O venv embutido é uma parte da biblioteca virtualenv
- O venv foi somente introduzido na biblioteca padrão em **2012** [3.3]
- Na [live de python #191](#) discutimos sobre ele!



```
pip install pacote  
pip uninstall pacote  
pip upgrade pacote
```

- Introdução do Pip Install Package (**pip**)
- Junto com o pip é introduzido o **requirements.txt**
- Instala, **desinstala** e atualiza
- Ambiente concreto, **dependências concretas!**





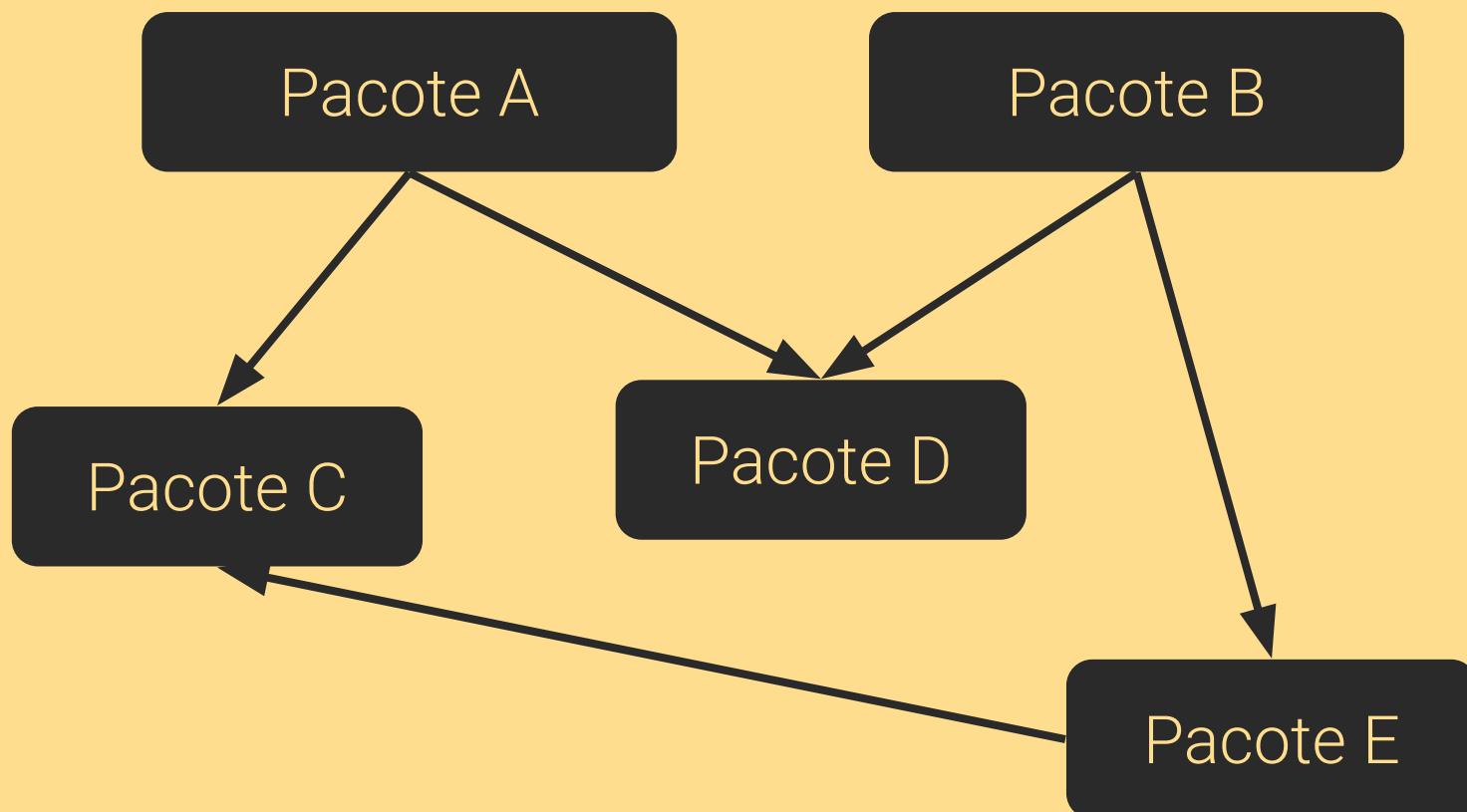
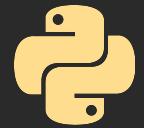
requirements.txt

O arquivo de requerimentos é uma forma de padronizar um **ambiente determinístico**. Com dependências **concretas**.

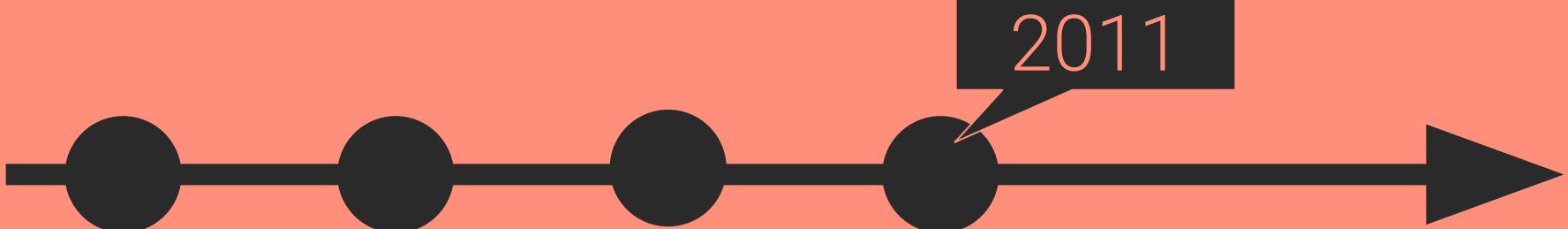
```
# requirements.txt
rich==13.7.0
rich-pixels==3.0.0
typer==0.9.0
inquirerpy==0.3.4
```

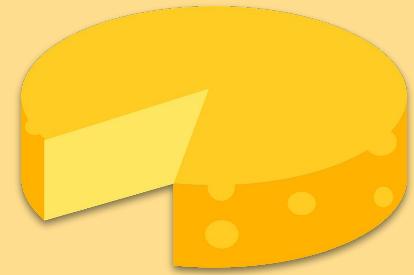
Especificação para a "pinagem" de versão foi introduzida na PEP-440 [2013]

Concreto vs Abstrato



- Ministério das instalações [PyPA] (Python Packaging Authority)
- Um grupo de pessoas voluntárias que cuida do:
 - setuptools
 - virtualenv
 - PyPI
 - ... *Ainda tem mais por vir*
- Lidar com a **complexidade** do ecossistema!

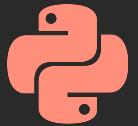




- O padrão **wheel** (extensão .whl)
- Padronização dos pacotes binários (**bdist**)
- Pacotes por:
 - sistema (windows, mac, linux, ...)
 - versão do python
 - plataforma (pypy, cpython, ...)
 - Exemplo Numpy
- PEP-427

2012 - 2014





Ponto importante sobre o wheel

- □ ×

```
{distribution}-{version}({build tag})?-{python tag}-{abi tag}-{platform tag}.whl
```

Onde os mais importantes agora são:

- distribution: o nome do pacote
- version: a versão do pacote
- python tag: Versão do python em que o pacote deve ser usado
- platform tag: tag para um sistema operacional específico. Para todos é `any`

As tags abi são um assunto a parte. Na [\[documentação do PyPA existe bastante insumo para isso\]](#)

Baixar arquivos

Baixe o arquivo para sua plataforma. Se você não tem certeza qual escolher, saiba mais sobre [instalação de pacotes](#).

Distribuição de Origem

[numpy-1.26.4.tar.gz](#) (15.8 MB [ver hashes](#))
Uploaded About 6 days ago. [source](#)

Built Distributions

- [numpy-1.26.4-pp39-pypy39_pp73-win_amd64.whl](#) (15.7 MB [ver hashes](#))
Uploaded About 6 days ago. [pp39](#)
- [numpy-1.26.4-p039-pypy39_pp73-manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#) (18.1 MB [ver hashes](#))
Uploaded About 6 days ago. [pp39](#)
- [numpy-1.26.4-p039-pypy39_pp73-macosx_10_9_x86_64.whl](#) (20.5 MB [ver hashes](#))
Uploaded About 6 days ago. [pp39](#)
- [numpy-1.26.4-cp312-cp312-win_amd64.whl](#) (15.5 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-win32.whl](#) (5.7 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-musllinux_1_1_x86_64.whl](#) (17.8 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-musllinux_1_1_aarch64.whl](#) (13.6 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#) (18.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-manylinux_2_17_aarch64.manylinux2014_aarch64.whl](#) (13.9 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-macosx_11_0_arm64.whl](#) (13.7 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp312-cp312-macosx_10_9_x86_64.whl](#) (20.3 MB [ver hashes](#))
Uploaded About 6 days ago. [cp312](#)
- [numpy-1.26.4-cp311-cp311-win_amd64.whl](#) (15.8 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-win32.whl](#) (6.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-musllinux_1_1_aarch64.whl](#) (18.1 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-manylinux_1_1_aarch64.whl](#) (13.9 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#) (18.3 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl](#) (14.2 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-macosx_11_0_arm64.whl](#) (14.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp311-cp311-macosx_10_9_x86_64.whl](#) (20.6 MB [ver hashes](#))
Uploaded About 6 days ago. [cp311](#)
- [numpy-1.26.4-cp310-cp310-win_amd64.whl](#) (15.8 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-win32.whl](#) (6.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-musllinux_1_1_x86_64.whl](#) (18.1 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-manylinux_1_1_aarch64.whl](#) (13.9 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#) (18.2 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl](#) (14.2 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-macosx_11_0_arm64.whl](#) (14.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp310-cp310-macosx_10_9_x86_64.whl](#) (20.6 MB [ver hashes](#))
Uploaded About 6 days ago. [cp310](#)
- [numpy-1.26.4-cp39-cp39-win_amd64.whl](#) (15.8 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-win32.whl](#) (6.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-musllinux_1_1_x86_64.whl](#) (18.1 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#) (18.2 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl](#) (14.2 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-macosx_11_0_arm64.whl](#) (14.0 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)
- [numpy-1.26.4-cp39-cp39-macosx_10_9_x86_64.whl](#) (20.6 MB [ver hashes](#))
Uploaded About 6 days ago. [cp39](#)

Exemplo distribuição do Numpy

pyproj
ect
.toml

Uma especificação!

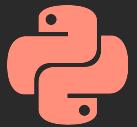


A especificação

O que temos até agora aprovado são 4 PEPs:

- 517 [2015-2017]: Backend e frontend de build
- 518 [2016]: Criação do `pyproject.toml`
- 621[2020]: Metadados
- 660 [2021]: Pacotes editáveis

PEP 518 - Um sistema mínimo para builds



- A ideia principal é mostrar como um pacote deve especificar suas próprias dependências de build. O que é necessário que o ambiente tenha, para que o build possa ser feito.
- Fica especificado que um arquivo chamado `pyproject.toml` será utilizado para esse fim.
- O arquivo no `toml` foi escolhido por ser bom de ler/escrever por humanos (diferente do JSON), flexível o suficiente (diferente do configparser), é originado por um padrão (também diferente do configparser) e não é complicado (como YAML).



TOML

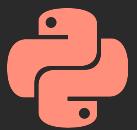
Implementa diversos tipos de dados, sendo o primordial para o pyproject
as **tables**:

```
# Comentário !
[table]
chave_0 = true # booleano
chave_1 = 'valor 1' # string
# array de strings
chave_2 = ['valor 1 no array', 'valor 2 no array']
chave_3 = 1 # inteiro
# Cria uma subtabela em uma linha
chave_4 = { nome = 'pyproject', extensao = '.toml'}

[table.subtable] # Cria uma subtabela
chave_0 = 1.0 # float

[table.subtable.subsubtable]
chave_0 = 2024-01-01
```

Equivalente a:



```
{  
    'table': {  
        'chave_0': True,  
        'chave_1': 'valor 1',  
        'chave_2': ['valor 1 no array', 'valor 2 no array'],  
        'chave_3': 1,  
        'chave_4': {'extensao': '.toml', 'nome': 'pyproject'},  
        'subtable': {  
            'chave_0': 1.0,  
            'subsubtable': {'chave_0': datetime.date(2024, 1, 1)}  
        }  
    }  
}
```



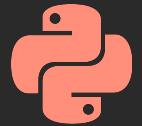
Especificação de build

A especificação se baseia em duas tables, uma para o sistema de build (**build-system**) e outra para ferramentas em geral (**tool**).

- A tabela referente ao sistema de builds tem uma única chave chamada **requires**, que será um array de todas as ferramentas que devem estar presentes no sistema no momento do build.

```
[build-system]
requires = ["setuptools", "wheel"]
# Suporta a PEP-508, ex: setuptools~=64.0
```

Desta forma, caso essas ferramentas não existam no ambiente, alguma ferramenta pode instalar (por padrão, o pip)



A tabela tool

Diferente da tabela de `build-system` que tem objetivos relacionados ao propósito da PEP, a tabela `tool` pode ser usada por qualquer ferramenta que relacionada ao seu projeto python, não somente a ferramentas de build.

A recomendação de uso é que cada ferramenta crie sua própria subtabela de tools segunda por `ferramenta`:

Alguns exemplos:

- Pytest: Biblioteca de testes
- Ruff: Linter e formatador
- towncrier: Gerador de changelogs (merece uma live?)
- mypy: Checador de tipos



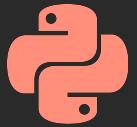
Um exemplo

```
[tool.ruff]
line-length = 79
exclude = ['.venv', 'migrations']

[tool.pytest.ini_options]
pythonpath = "."

[tool.taskipy.tasks]
lint = 'ruff . && blue --check . --diff && isort --check . --diff'
format = 'blue . && isort .'
run = 'uvicorn fast_zero.app:app --reload'
pre_test = 'task lint'
test = 'pytest -s -x --cov=fast_zero -vv'
post_test = 'coverage html'
```

Ferramentas de build + tool



As ferramentas de build também usam esses campos:

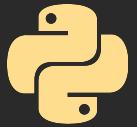
- poetry
- pdm
- hatch
- rye
- setuptools
- flit
- ...

```
[tool.hatch.envs.test]
dependencies = [
    "pytest"
]

[tool.setuptools.package-data]
"_pytest" = ["py.typed"]
"pytest" = ["py.typed"]

[tool.poetry.dependencies]
python = "3.11.*"
fastapi = "^0.109.0"
uvicorn = "^0.25.0"
```

PEP 621 - Metatados para o sistema de build



Embora um sistema mínimo de build tenha sido criado e bem utilizado por ferramentas. Alguns pontos do modelo "agnóstico" ainda não estavam sendo levado em consideração. Os metadados do projeto.

Os **backends** não tinham um padrão definido para o `nome` do pacote, sua `versão`, seus `scripts`, suas `dependencias`, etc. O que tornava a migração entre backends um grande trabalho e diminuía a facilidade "pregada" pela PEP-518.

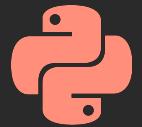
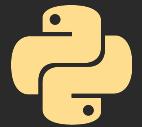


Tabela [project]

A PEP especifica uma nova tabela opcional no `pyproject.toml` chamada **project**. Que agrupa todos os metadados sem depender de uma tabela [tool.ferramenta]

A especificação diz que os valores atrelados as chaves podem ser de dois tipos:

- **Estaticos**: Metadados especificados no próprio arquivos e que não podem ser alterados
- **Dinâmicos**: Metadados marcados em um array com a chave `dynamic`, que serão fornecidos pelo build-system



Chaves obrigatórias

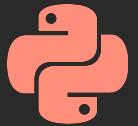
Somente as chaves **name** e **version** são consideradas obrigatórias

– □ ×

```
[project]
name = 'package'
version = '0.1.0'
```

ou

```
[project]
name = 'package'
dynamic = ['version']
```



Chaves opcionais

Entre as chaves opcionais estão uma grande variedade de chaves, como:

- **description**: String de descrição do pacote
- **reame**: String com o caminho do arquivo `readme.md`
- **license**: Subtabela indicando o arquivo
 - `'{file = 'LICENSE.txt'})`
- **authors**: Array de subtabelas inline
 - `'[{'name = 'duduzinho', email = 'test@mail'}]'`
- **manteiners**: Array de subtabelas inline
 - `'[{'name = 'duduzinho', email = 'test@mail'}]'`

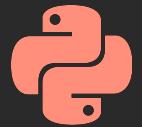
> As definições atualizadas de metadados podem ser encontradas aqui:
<https://packaging.python.org/en/latest/specifications/pyproject-toml/#declaring-project-metadata-the-project-table>



Dependências

Nas chaves opcionais eu gostaria de destacar o campo que especifica as dependências do projeto. Um array listando todos os pacotes que nosso pacote depende (uma espécie de requirements):

```
[project]
name = 'spam'
version = '0.1.0'
dependencies = ['httpx', 'trio', 'anyio']
```

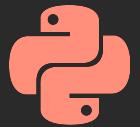


A magia acontecendo!

Agora, com um **build-system** + **project**. Podemos usar o pip para instalar nosso pacote local, baixar e instalar as dependências.

```
[project]
name = 'spam'
version = '0.1.0'
dependencies = [
    'typer',
    'rich-pixels<3.0.0',
    'inquirerpy',
    'pillow',
]

[build-system]
requires = ["setuptools"]
```



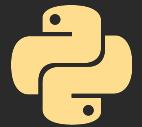
A magia acontecendo!

Agora, com um **build-system** + **project**. Podemos usar o pip para instalar nosso pacote local, baixar e instalar as dependências.

```
[project]
name = 'spam'
version = '0.1.0'
dependencies = [
    'typer',
    'rich-pixels<3.0.0',
    'inquirerpy',
    'pillow',
]
```

```
[build-system]
requires = ["setuptools"]
```

```
pip install .
Processing /home/dunossauro/live_pyproject/pacote_de_exemplo
Installing build dependencies ... done
Getting requirements to build wheel ... done
Installing backend dependencies ... done
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: typer in ./venv/lib/python3.12/site-
packages (from spam==0.1.0) (0.9.0)
```



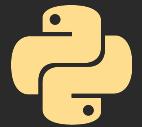
Subtabelas de project

Dentro da tabela `project` você pode definir diversas outras tabelas, vou destacar as que considero mais interessantes agora:

- **optional-dependencies**: Uma tabela que cria grupos de dependências. Como dependências de testes, documentação e etc...
- **scripts**: Executáveis do pacote.

A subtabela de dependências opcionais:

```
[project.optional-dependencies]
dev = ['ruff', 'mypy']
test = ['pytest', 'pytest-cov']
doc = ['mkdocs']
```



Subtabelas de project

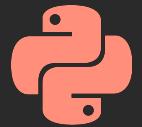
Dentro da tabela `project` você pode definir diversas outras tabelas, vou destacar as que considero mais interessantes agora:

- **optional-dependencies**: Uma tabela que cria grupos de dependências. Como dependências de testes, documentação e etc...
- **scripts**: Executáveis do pacote.

A subtabela de dependências opcionais:

```
[project.optional-dependencies]
dev = ['ruff', 'mypy']
test = ['pytest', 'pytest-cov']
doc = ['mkdocs']
```

```
pip install .[dev]
pip install .[test]
pip install .[doc]
```



Subtabela de scripts

A subtabela de `scripts` também é interessante para criar CLIs ou entradas para GUIs diretamente do pacote:

- □ ×

```
[project.gui-scripts]  
spam-gui = "spam.main:gui"
```

```
[project.scripts]  
spam-cli = "spam.main:cli"
```

PEP 517 - O sistema de builds

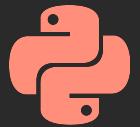


O objetivo de definir um formato de build independente das ferramentas já existentes. Como `buildutils` e `setuptools`, definidas pelo `distutils-sig`. Se você quiser usar outra coisa, isso deve ser fácil de fazer e também deve ser fácil para novas ferramentas serem criadas.

Desde a PEP-427 (2012) o formato padrão dos binários é o `wheel`. Isso faz com que todos os pacotes distribuídos sejam únicos.

O sistema de builds agora é dividido em duas partes:

- Backend
- Frontend



O Backend

O **backend** é responsável por compilar um pacote para distribuição em **sdist** e **bdist**.

Para fazer isso as ferramentas de backend devem implementar algumas chamadas (hooks / funções) obrigatórias:

- **build_wheel**: Função que cria o pacote binário especificado na PEP-427
- **build_sdist**: Função que faz a compactação da biblioteca em um `tar`

> Existem outros opcionais, mas acredito que a leitura da PEP-517 pode aprofundar mais. A PEP-660 (2021) expande novos hooks opcionais para que os wheels sejam editáveis.

Backend



O sistema de builds deve implementar um módulo com os hooks, esse hooks devem ser implementados na chave **build-backend**. Algo como:

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

```
# hatchling/build.py
def build_wheel(
    wheel_directory,
    config_settings=None,
    metadata_directory=None
):
    ...
    ...

def build_sdist(
    sdist_directory,
    config_settings=None
):
    ...
    ...
```



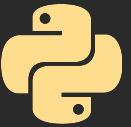
Backends buildando backends

Um problema comum ao criar um backend é que para que seu build seja feito, ele precisa conseguir chamar a si mesmo. Para isso foi criada uma nova chave na tabela **backend-path**. Para especificar onde o módulo que contém os hooks estão:

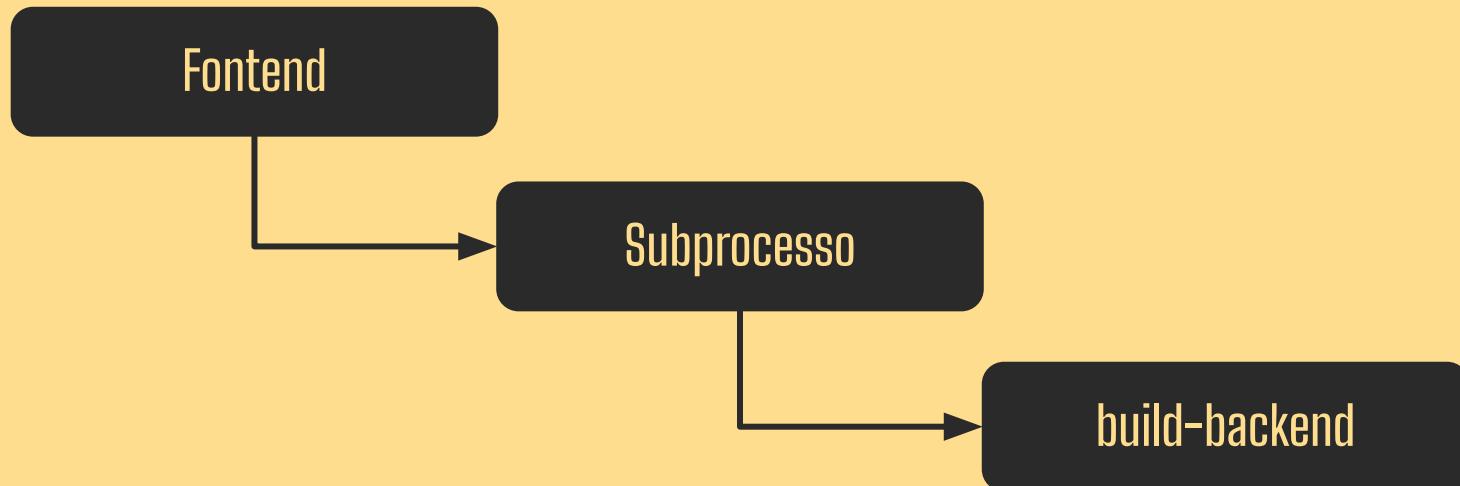
- □ ×

```
[build-system]
requires = []
build-backend = "backend.back"
backend-path = ["."]
```

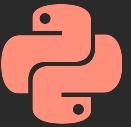
Frontends



O frontend é responsável por configurar o ambiente de build do pacote. Não é exigido nenhum ambiente em específico, a recomendação é que se use o **virtualenv**. Aí ideia é que o hooks sejam chamados em subprocessos (**subprocess**) e que os pacotes sejam pelo backend.



Frontends

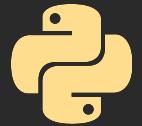


Várias ferramentas podem agir como frontend e o objetivo é que sejam agnósticas nas chamadas. Pois os backends devem seguir a especificação.

Duas ferramentas em particular são importantes nesse contexto. O **build**, criado pelo PyPA para ser uma ferramenta genérica para o `pyproject.toml` e o **pip**.

– □ ×

```
pip install .
pip install build
python -m build
```



Relação entre back e front

- Poetry (front) -> Poetry code [masonry] (back)
- Build (front) -> setuptools (back)
- Hatch (front) -> Hatcling (back)
- pip (front)
- rye (front)
- ...

O objetivo final é que qualquer front consiga chamar qualquer back!

Futu ro?

Nem tudo está
pronto!



Algumas coisas faltando

Embora tenhamos andado um bastante pelo sistema de builds e instalação de pacotes, ainda existem pontos em aberto:

1. Pip instalar as dependências sem chamar o backend
 - a. <https://github.com/pypa/pip/issues/11440>
2. Criar instalações deterministicas (como requirements)
 - a. PEP-665 (rejeitada)
 - b. PEP ainda sem número
 - i. <https://github.com/FRidh/peps/blob/pep-0697/pep-0697.rst>



apoia.se/livedepython



pix.dunossauro@gmail.com



patreon.com/dunossauro



Ajude o projeto <3

