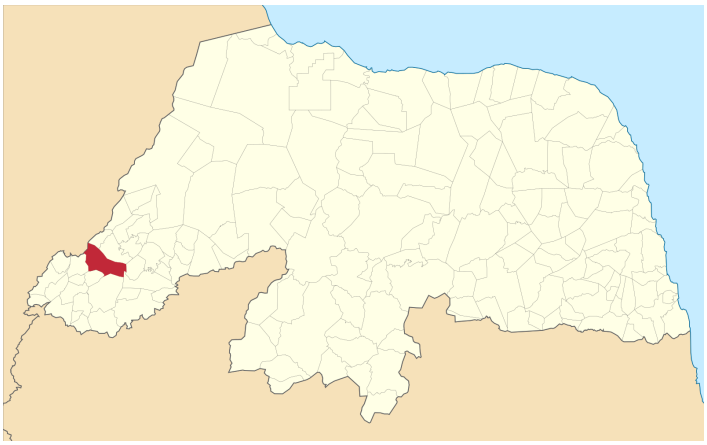
Indo além com Jupyter Notebooks

Bem vindo! Esse tutorial irá mostrar como utilizar Jupyter notebooks para publicar pacotes Python & mais

Olá, sou o Ítalo Epifânio



- Norte-rio-grandense da trombinha do elefante
- Cientista da Computação*
- RnD Python Developer
- Desenvolvedor Open Source

- Contribuidor da comunidade Python (Grupy RN)

Sumário

Esse tutorial irá:

- Introduzir os principais conceitos para publicar um pacote Python
- Discutir o sistema de pacotes Python e como distribui-los
- Explicar o conceito de programação letrada
- Mostrar como utilizar Jupyter notebook e Jupyter lab
- Introduzir a biblioteca Nbdev
- Explicar boas práticas de desenvolvimento em notebooks

Sumário

- Mostrar algumas ferramentas para assegurar qualidade de código
- Discutir como rodar testes em software
- Mostrar como e onde publicar
- Introduzir o uso de Github Actions e CI/CD
- Bônus (se o tempo permitir):
  - Falar sobre pacotes python (passado e futuro)
  - Introduzir como construir UI para Jupyter notebooks
  - Mostrar como customizar documentações utilizando Quarto

- Demonstrar como utilizar ChatGPT em notebooks

Pacotes Python

Vamos introduzir/revisar alguns conceitos chaves para entender pacotes python e sua estrutura

Módulos

> Módulos são arquivos que contem definições e declarações Python Python Docs

Qualquer arquivo `.py` pode ser considerado um módulo

```
%%writefile module.py

def hello(name):
    print(f'hello, {name}')
```

```
import module

module.hello("audience!")
```

Módulo

Módulos também podem ser executados como scripts python:

```
%%writefile module.py

def hello(name):
    print(f'hello, {name}')

if __name__ == "__main__":
    import sys
    hello(sys.argv[1])
```

```
! python module.py 'Maria'
```

> **Note**
>
> O condicional `if __name__ == "__main__"` verifica que a função `hello` será executada somente quando o módulo estiver sendo executado como o arquivo principal (`main`)

Sistema de busca de módulos

O interpretador Python primeiro busca:

- Nos módulos built-in (listados em `sys.builtin_module_names`)
- Nos arquivos `.py` do diretório atual (`sys.path`)

Pacotes Python

> É uma forma de estruturar namespaces no Python utilizando nomes de módulos separados por pontos Python Docs

Pacotes podem ser entendidos como uma coleção de módulos. Verifique a seguinte estrutura:

```
somefolder/
    package/
        __init__.py
        module1.py
        subpackage/
            module2.py
```

Sistema de busca por pacote

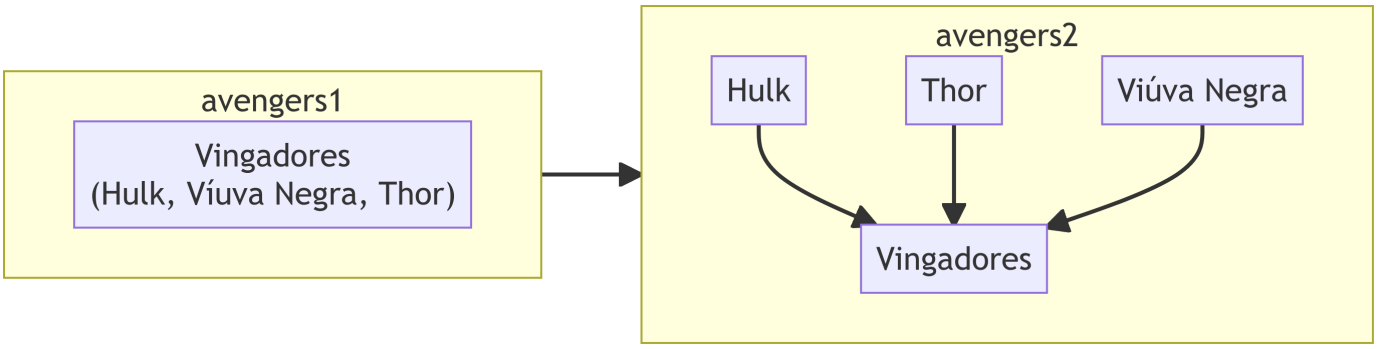
Quando importa-se pacotes, o interpretador Python busca por diretórios listados em `sys.path`

Por que utilizar pacotes?

Vamos discutir os prós e contras de utilizar pacotes

Por que utilizar pacotes?

Vamos pensar no filme "Vingadores" como um grande pacote com diversos super heróis (Hulk, Viúva Negra, Thor, etc).



> **Note**
>
> Exemplo da palestra Arquitetura Modular com pacotes Python apresentado na Python Brazil, 2022

Para utilizar super heróis em diversas linhas do tempo a duplicação de código seria necessária no avengers1, o que aumentaria a complexidade do código. A ideia de criar pacotes para cada super-herói permite combinar eles em diferentes filmes.

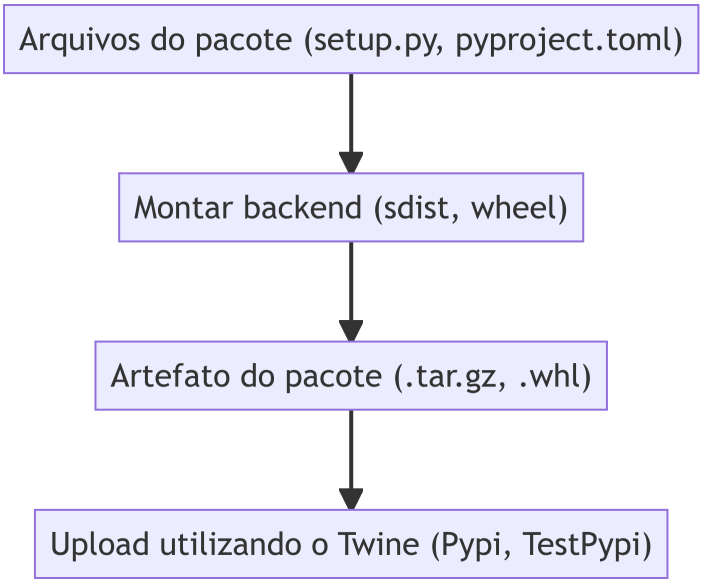
Prós e contras

- Aumenta reuso de código
- Diminui acoplamento
- Facilita com que outros desenvolvedores usem o código
- Divide a responsabilidade entre times
- Pode facilitar a manutenção

- Gestão de dependencia pode ser custosa
- É difícil garantir a segurança de códigos de terceiros
- Pacotes podem ter mais funcionalidades que as necessárias

Como publicar?

Como publicar pacotes no PyPI e então baixar?



Pypi

Python Package Index

Ferramental

- pip é o oficial e mais popular gerenciador de pacotes Python
- PyPI é o catalogo de onde o pip baixa os arquivos
- Test PyPI um catalogo separado para testes

> **Note**
>
> Crie sua conta em https://test.pypi.org, vamos utilizar ele para publicar um pacote

Outros repositórios Python

Existem outros respositórios além do PyPI e Test PyPI

- Pypiserver local ou auto-hospedagem
- JFrog auto-hospedagem ou cloud, solução completa de cloud
- Code artifacts repositório da AWS
- Artifact registry repositório da Google

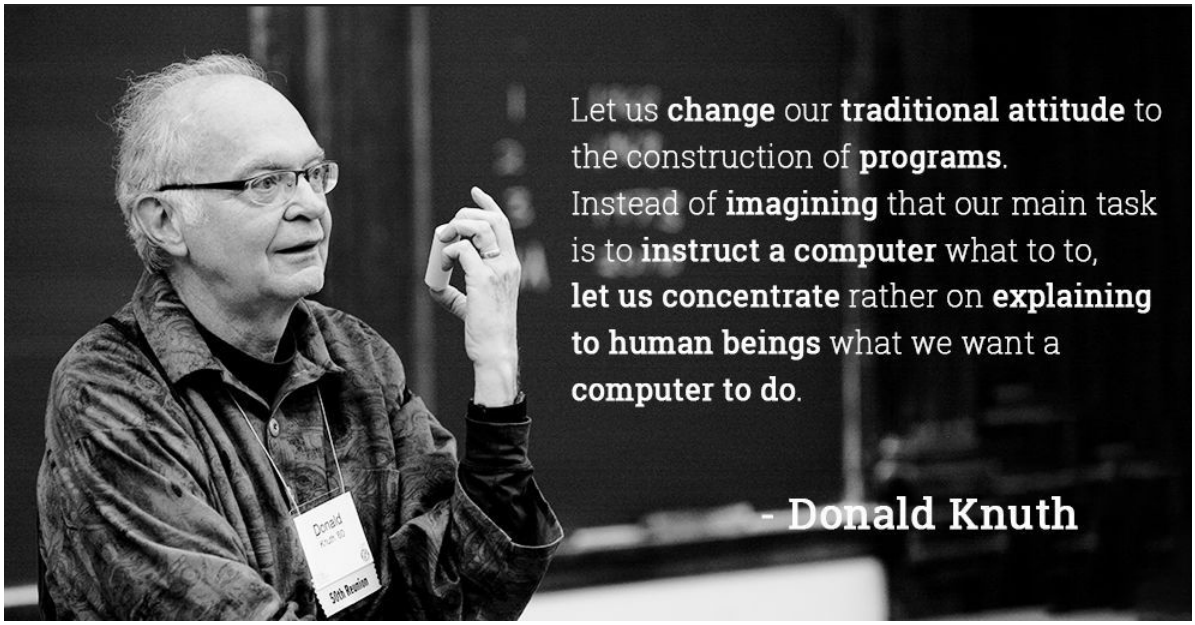
- Gitlab package registry repositório do Gitlab

Programação Letrada

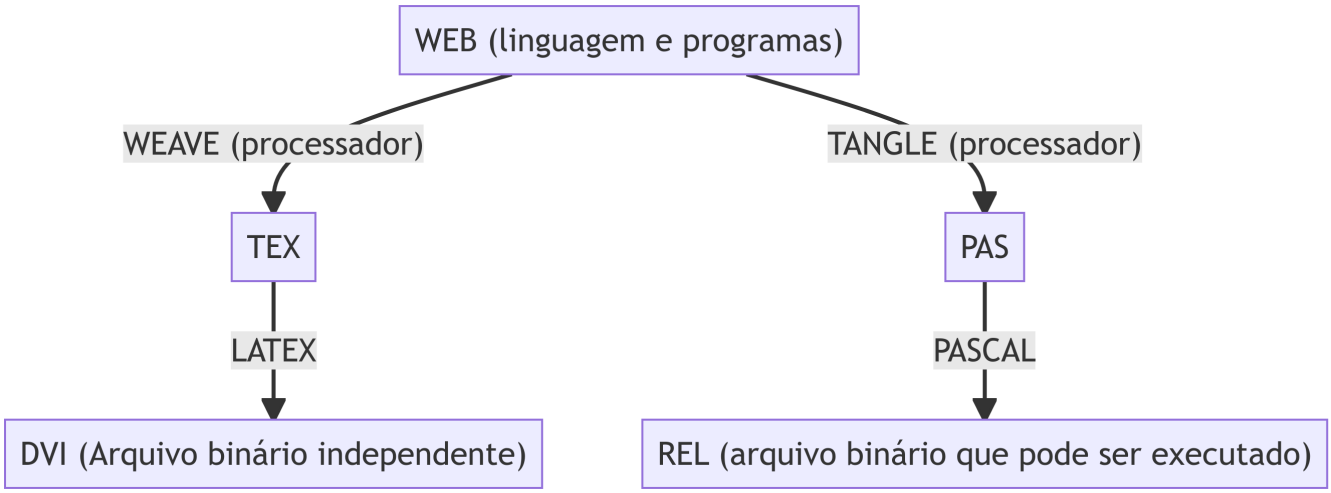
Paradigma em que você conda uma história com seu código

Programação Letrada

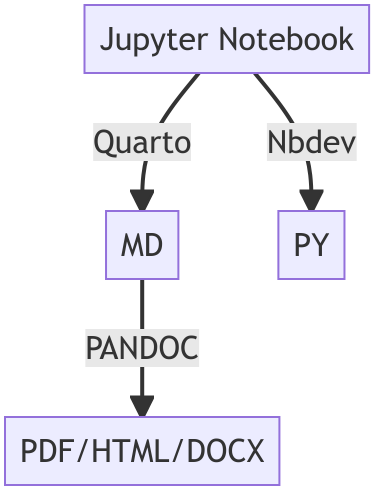
- Paradigma de programação
- Haskell `.lhs` vs `.hs`
- Sua utilização aumentou nos últimos anos
- Jupyter notebooks, R Studio



Programação Letrada



Literate programming



Introdução a jupyter notebook

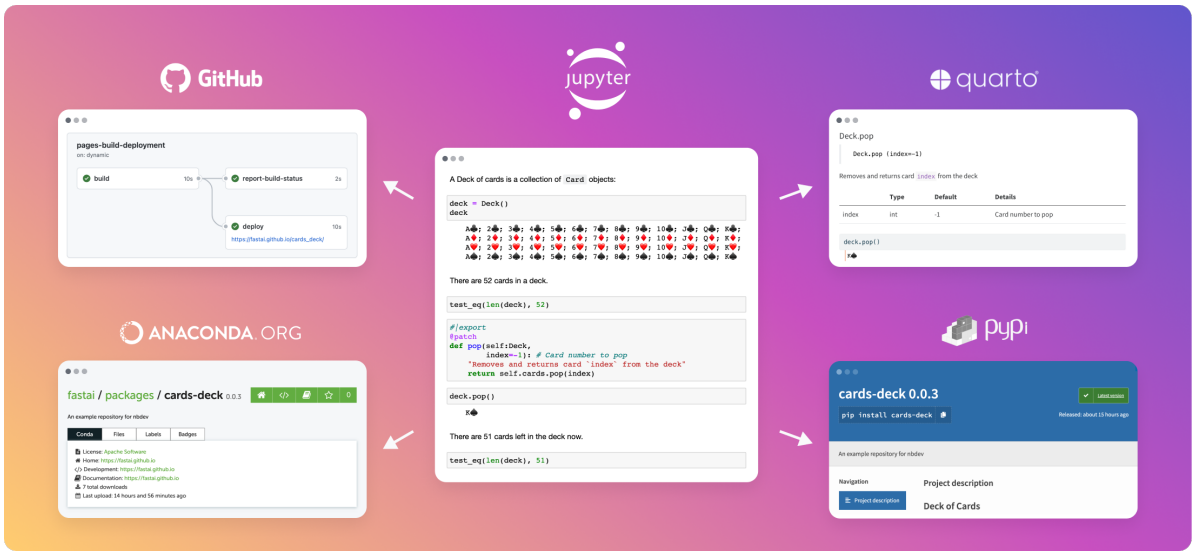
- Crie um repositório Github vazio (não adicione readme, .gitignore ou licença)
- Clone o repositório localmente e acesse sua raiz
- Crie seu ambiente virtual `python -m venv venv`
- Ative-o utilizando `source venv/bin/activate`
- Instale as dependências necessárias `pip install nbdev notebook twine`
- Execute `jupyter-notebook`

> UI
>
> - Jupyter Lab é uma das ferramentas mais populares
> - Instale a biblioteca utilizando `pip install jupyterlab` e execute `jupyter-lab` para ver sua interface
>
> Existem outras interfaces, como o plugin VSCode que renderiza notebooks, mas eu recomendo utilizar notebooks clássicos nesse tutorial por ter uma interface mais simplificada

Nbdev

Utilizando Jupyter notebooks para publicar pacotes Python

Nbdev



Setup

- Execute `nbdev_new --lib_name {username}_pacote`
- Verifique as informações em `settings.ini`
- Realize o commit das alterações: `git add . && git commit -m "Commit inicial" && git push origin main`
- Instale o pacote localmente utilizando `pip install -e .[dev]`
- Execute `nbdev_install_hooks`
- Execute `nbdev_install_quarto`

> **Note**
>
> Vamos nos familiarizar com a estrutura nbdev (settings.ini, nbs/, setup.py)
> Execute `nbdev_help` no seu terminal para verificar todos os comandos disponíveis

Nbdev comentários mágicos

- `#| hide`
- `#| export`
- `#| exporti`
- `#| exec_doc`
- `#| code-fold`
- `#| default_exp core`
- `#| eval: false`

Nbdev awesome projects

- FastAI
- Ipyannotator
- TSAI
- FastKafta
- Number Blog
- UPIT
- AskAI
- Streamlit Jupyter
- Banet

Boas práticas

Melhores práticas com Jupyter notebook e nbdev

Tipos de Jupyter notebook

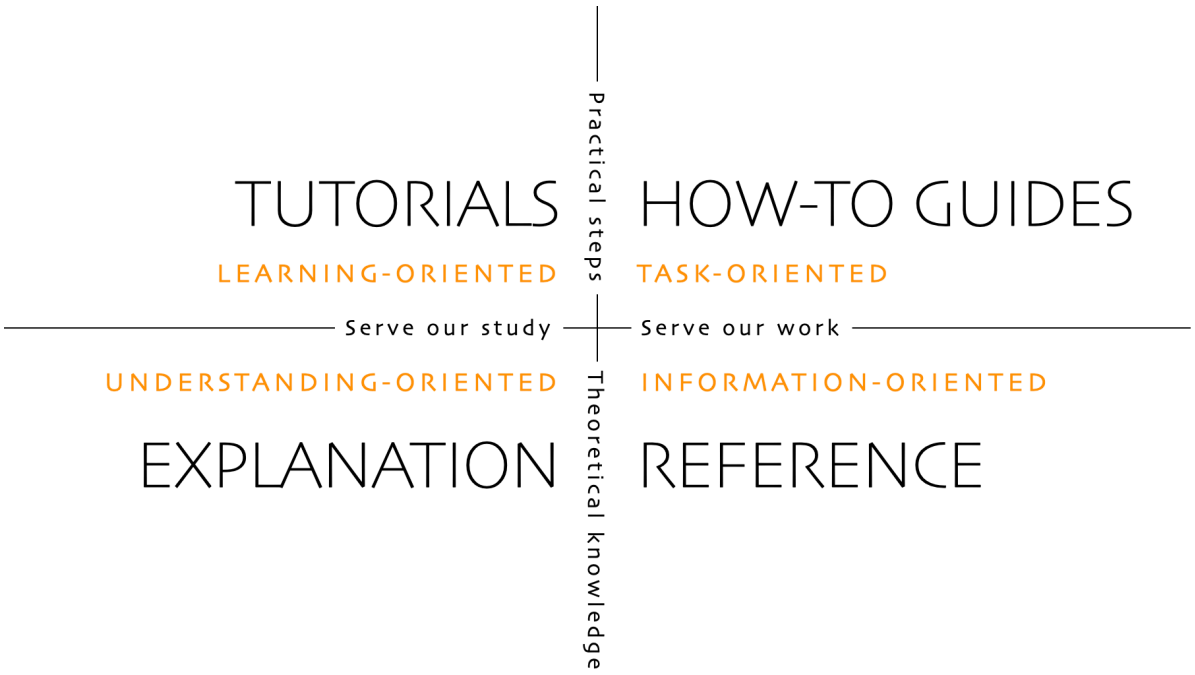


Figure 1: Saiba que tipo de notebook você está escrevendo (Sistema Diataxis)

Bom título e subtítulo

Existem duas formas de fazer isso, a primeira com markdown:

```
# Meu título H1

> Minha descrição
```

Ou usando a sintaxe Quarto:

```
---
title: "Meu título"
description: "Minha descrição"
```

```
---
```

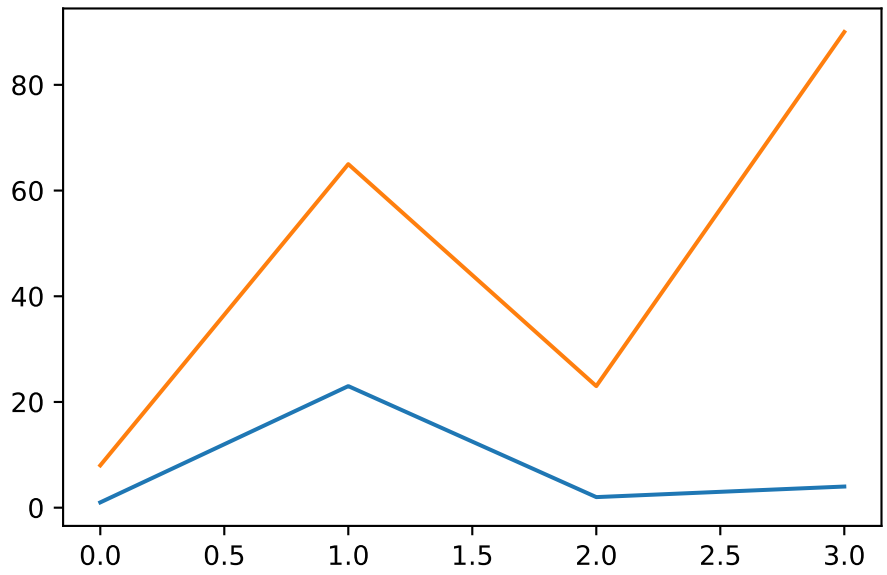
Mude o texto de acordo com o tipo de notebook

- Referência: Inicie com uma descrição geral do componente, utilize links para facilitar navegação
- Tutorial e guias: Descreva ao leitor o que ele vai aprender e como. Seja objetivo
- Explicação: Breve explicação do tópico é suficiente para guiar o leitor durante a leitura

Use visualizações

Jupyter notebooks são bastante interativos. Use e abuse de images, videos e audios

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.plot([8,65,23,90])
plt.show()
```



Mantenha suas células pequenas

- Não existem regras para o tamanho das células, mas seja razoavel
- Use @patch `from fastcore.basics import patch`

- Melhore a testabilidade do notebook adicionando pequenos testes abaixo de cada célula de código

Jupyter notebooks

- Mantenha seus imports no topo do notebook
- Não import bibliotecas e execute código python na mesma célula
- Evite ordens de execução ambíguas
- Use células para experimentar o código desenvolvido

Documentação de parâmetros

Nbdev tem duas formas de documentação, a clássica (estilo numpy), como se segue:

```
def add_np(a, b=0):
    """The sum of two numbers.

    Parameters
    ----------
    a : int
        the 1st number to add
    b : int
        the 2nd number to add (default: 0)
    """
    return a + b
```

Documentação de parâmetros

```
from nbdev.showdoc import show_doc

def add_np(a, b=0):
    """The sum of two numbers.

    Parameters
    ----------
    a : int
        the 1st number to add
    b : int
        the 2nd number to add (default: 0)
```

```
    """
    return a + b

show_doc(add_np)
```

---

add_np

```
add_np (a, b=0)
```

The sum of two numbers.

	Type	Default	Details
a	int		the 1st number to add
b	int	0	the 2nd number to add (default: 0)

Doc parameters

E a abordagem nbdev chamada de “docments”

```
def add(
    a: int,  # the 1st number to add
    b=0,  # the 2nd number to add
):
    "The sum of two numbers."
    return a + b
```

```
from nbdev.showdoc import show_doc

def add(
    a: int,  # the 1st number to add
    b=0,  # the 2nd number to add
):
    "The sum of two numbers."
    return a + b

show_doc(add)
```

---

add

```
add (a:int, b=0)
```

The sum of two numbers.

	Type	Default	Details
a	int		the 1st number to add
b	int	0	the 2nd number to add

Escreva testes

- A palavra chave `assert` pode ser utilizada para testes
- `nbdev_test`

```
assert add(1, 1) == 2
```

Dependências

Evite “dependency hell”

Dependency Hell

- Quando um software cresce há uma tendência que mais bibliotecas sejam adicionadas
- Pacotes são massa: adicionam funcionalidades, evitam/corrigem erros
- Manutenção e atualização de pacotes pode ser desafiador
- Ex. Dependência 1 espera Python 3.7 e dependência 2 espera Python 3.9

Especifique suas dependências

- ~= para releases específicas
- == para fixar uma versão
- != para excluir uma versão
- <=, >=, <, > para incluir um range de versões

Releases

- Versionamento semântico é um conjunto de regras que pode evitar dependency hell
- Para isso a biblioteca DEVE declarar uma API pública compreensível
- O versionamento DEVE usar o formato X.Y.Z onde X, Y, Z são inteiros não negativos e cada elemento deve ser aumentando numericamente
- The version number MUST use the X.Y.Z format where X, Y, Z are
- MAJOR.MINOR.PATCH

Mantenha um changelog

- Versionamento semântico é bastante didático mas não é legível para humanos
- Changelog mantém uma ordem cronológica de modificações notáveis a cada versão do projeto
- Melhora a transparência
- Entendimento geral da direção do projeto por desenvolvedores ou não desenvolvedores
- Mantém uma lista de bugs resolvidos

Garantia de qualidade

Melhore seu código utilizando ferramentas automatizadas

NbQA

Existem diversas ferramentas de análise de código que:

- Evitam erros e más práticas de código (code smells)
- Definem estilo de código
- Melhoram a leitura
- Medem a qualidade de código

NbQA

Ferramentas populares:

- Autopep8
- Black
- Flake8
- MyPy

Todas essas (e algumas outras) podem ser executadas utilizando NbQA

```
nbqa <tool-name> <tool-params>
```

Autopep8

> Formatador de código que segue a PEP 8

Muito útil para limpar espaços em branco e realizar formação de código em notebooks

Execute o seguinte comando em seu terminal:

```
! nbqa autopep8 nbs/*.ipynb --in-place
```

Publicando

Nbdev permite que publiquemos pacotes python utilizando sua CLI

Conda and PyPI

- `nbdev_pypi`, `nbdev_conda`, `nbdev_release_both`
- Menos documentado mas também permite publicar no Test PyPI (`nbdev_pypi --repository testpypi`)
- Vamos publicar nosso pacote `nbdev_pypi --repository testpypi`

Chaves secretas

O sistema de pacote python usam o arquivo .pypirc para definir os repositórios:

```
[distutils]
index-servers = testpypi

[testpypi]
repository=https://test.pypi.org/legacy/
username=<your-username>
password=<your-password>
```

Chaves secretas

Username e senhas não são encorrajadas, considere gerar um token para seu pacote:

```
[distutils]
index-servers = testpypi

[testpypi]
repository=https://test.pypi.org/legacy/
username = __token__
password = <PyPI token>
```

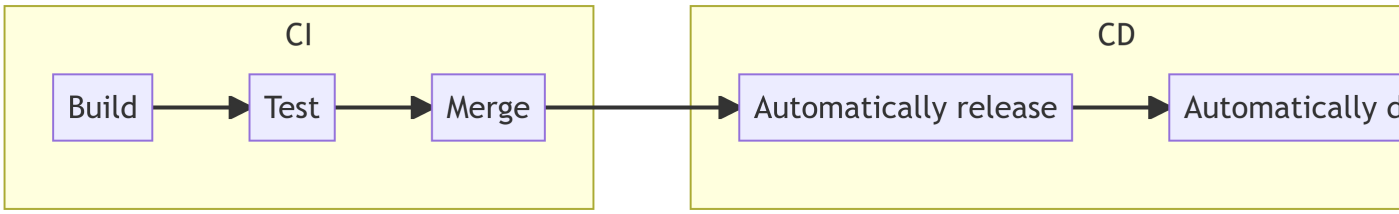
CI/CD

Vamos executar testes, ferramentas de QA e publicar nosso pacote utilizando Github Actions

Integração Contínua/Entrega Contínua

- Conceito DevOps que recentemente se tornou mais acessível a todos os desenvolvedores
- Metodologia de frequentemente entregar novas versões de apps a clientes
- Automatiza etapas do processo de entrega de novas versões
- Garante segurança nessa entrega

Integração Contínua/Entrega Contínua



Ferramentas

- Github Actions
- Gitlab CI/CD
- Jenkins
- Circle CI

E várias outras

Github Actions

- Fácil de usar
- Grátis até 2000 minutos/mês
- Maioria das ferramentas tem uma integração

Github actions conceitos principains

- *Events*: É uma atividade específica do seu repositório Github que pode iniciar alguma ação. Por exemplo, abrir um PR, enviar um commit, etc
- *Jobs*: Sequencia de passos que vão ser executados via shell script ou action. Jobs podem ser executados em paralelo ou sequencialmente
- *Action*: Aplicação customizada que usa a plataforma Github Action, normalmente automatiza alguma tarefa repetitiva como configurar um ambiente (ex. Python) ou gerenciar dependências mais complexas
- *Runner*: Servidor que executa os jobs. Cada runner executa um job de cada vez

Github secrets & Test Pypi token

- Vamos criar um token para o nosso pacote

> **Note**
>
> Na interface Test PyPI acesse `Account Settings >> Api Tokens >> Add Api Token`

- Na interface do seu repositório Github acesse: `Settings >> Actions (at security tab) >> New repository secret` and add the token with the name `TEST_PYPI_API_TOKEN`

Workflows

Adicionando arquivos `.github/worflows/*.yaml`

- Vá para https://l1nq.com/nb-pyne-2023
- Copie as pastas `scripts`, `.github` para seu repositório

- Copie o arquivo `.flake8` para o seu repositório

Publishing using CI

> **Note**
>
> - Atualize a versão da biblioteca `nbdev_bump_version`
> - Adicione `dev_requirements = nbdev autopep8 flake8 mypy` no arquivo `setting.ini`
> - Realize um commit e dê push nas suas mudanças
> - Verifique se os teste e lint estão passando
> - Crie uma nova release na interface do Github

Mais teoria ou prática?

Você que decide

História dos pacotes python

Vamos discutir o passado e futuro dos pacotes Python

História

- Python 1 (1998-2000) não tinha um gerenciador de pacote
- Distutils foi adicionado em Python 1.6 utilizando setup.py
- Em 2003 setuptools foi introduzido como melhoria ao distutils
- Em 2004 easy_install foi desenvolvido para ser utilizado junto ao setuptools
- Em 2008 a PyPA (Python Packaging Authority) foi fundada

History

- Em 2011 `pip` se tornou o gerenciador de pacotes padrão
- Em 2013 o formato `wheel` foi introduzido
- Em 2017 o pacote `flit` introduziu `pyproject.toml`
- Em 2020 a PEP 621 fez `pyproject.toml` se tornar o padrão de configuração de pacotes

Quarto e Estilização

Quarto é uma ferramenta open source que permite que você crie conteúdo utilizando Python, R, Julia e Observable. Nbdev <3 Quarto

Doc preview

Mudar de portas quando renderizando a documentação pode ser chato. Você pode fixar uma porta e evitar abrir novas abas:

```
project:
  preview:
    port: 3000
    browser: false
```

Navegação da documentação

Fácil customização da barra de navegação

```
website:
  navbar:
    background: primary
      search: true
      collapse-below: lg
      left:
        - text: "My page"
          href: index.ipynb
      right:
        - icon: github
          href: "https://github.com/user/project"
```

Google analytics

```
website:
  google-analytics: "UA-XXXXXXXX"
  cookie-consent: true
```

Dark mode

```
format:
  html:
    theme:
      light: flatly
      dark: darkly
```

Page navigation

Se seu projeto precisa de navegação contínua

```
website:
  page-navigation: true
```

Modo leitura

```
website:
  reader-mode: true
```

Playground

Já publicamos nosso pacote, agora vamos descobrir mais dos super poderes dos Jupyter notebooks

Ipywidgets

Vamos discutir interfaces, voila, tendencias de desenvolvimento de data science

> Tip
>
> Executar `03_ui_for_jupyter_notebook.ipynb` usando `voila 03_ui_for_jupyter_notebook.ipynb`

ChatGPT

Vamos utilizar o ChatGPT em nossos nbs

Conclusão do tutorial

- Jupyter notebooks são massa para prototipação rápida
- Código, documentação e testes podem ser utilizados para contar uma história do seu código
- Jupyter notebooks & nbdev permitem utilizar o paradigma da programação letrada
- Publicar pacotes com nbdev é fácil, rápido e eficiente
- Bibliotecas visuais se beneficiam muito de nbdev