|  |  |  |  |
| --- | --- | --- | --- |
| Desenho de personagem de desenhos animados com texto preto sobre fundo branco  Descrição gerada automaticamente com confiança média | Ano Letivo:  20241 | Turno:  Noturno | Local:  A definir |
| Curso:  ENGENHARIA DE SOFTWARE | | Turma:  5ª Fase |
| Disciplina:  ENGENHARIA DE DADOS | | Professor(a):  JORGE LUIZ DA SILVA | |

**Python para Engenharia de Dados**

**VISÃO GERAL**

|  |  |
| --- | --- |
| Gerenciador de versões do Python | **Pyenv** |
| Gerenciador de pacotes, bibliotecas Python | **PipX** |
| Gerenciador de pacotes, bibliotecas e ambientes virtuais | **Poetry** |
| Versionamento de código | **Git** |
| Repositório de código fonte com controle de versão Git | **Github** |
| Git Ignore .gitignore | **Ignr** |
| Estrutura de pastas e arquivos para projetos Python | **Padrão de projetos** |
| Framework para geração de testes no Python | **Pytest** |
| Bibliotecas para padrões de código | **Docstrings, Blue, Isort, pip-audit** |
| Documentação web padrão Markdown | **Mkdocs** |
| Automatizador de tarefas e atalho para comandos complexos | **Taskipy** |

**PYENV**

Pyenv é uma ferramenta usada para gerenciar várias versões do Python em um sistema operacional. Ele permite que você altere facilmente a versão global do Python utilizada em seu sistema, bem como a versão específica (local) de cada projeto. É utilizada para garantir a compatibilidade com vários projetos que possam exigir diferentes versões do Python.

O que o pyenv faz...

* Permite alterar a versão global do Python por usuário.
* Fornece suporte para versões Python por projeto.
* Permite substituir a versão do Python através de uma variável de ambiente.

**LINK**

<https://github.com/pyenv/pyenv>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/pyenv-win/pyenv-win/master/pyenv-win/install-pyenv-win.ps1" -OutFile "./install-pyenv-win.ps1"; &"./install-pyenv-win.ps1"

Fonte: <https://github.com/pyenv-win/pyenv-win>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **pyenv** | Lista todas as opções disponíveis para o comando pyenv (help). |
| **pyenv --version** | Verifica a versão do Pyenv instalado. |
| **pyenv install --list** | Mostra todas as versões do Python disponíveis para instalação. |
| **pyenv install <versão>** | Instala uma versão do Python. Ex.: **pyenv install 3.12.1** |
| **pyenv versions** | Lista todas as versões do Python instaladas. |
| **pyenv global <versão>** | Define uma versão do Python como global do ambiente. Ex.: **pyenv global 3.12.1** |
| **pyenv local <versão>** | Define uma versão do Python dentro de uma pasta específica. Ex.: **pyenv local 3.12.1** |

**PIPX**

O pipx é uma ferramenta utilizada em Python para instalar e gerenciar pacotes Python globais de forma isolada em ambientes virtuais, permitindo que os pacotes sejam executados como programas independentes.

O que o pipx faz...

* Instalação global isolada.
* Gestão simplificada.
* Execução de pacotes como aplicativos.

**LINK**

<https://pipx.pypa.io/stable/>

**COMO INSTALAR (Windows):**

1. Abrir o Shell (Prompt de Comando) no Windows.
2. Executar o comando:

pip install pipx

Fonte: <https://pipx.pypa.io/stable/>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **pipx** | Lista todas as opções disponíveis para o comando pipx (help). |
| **pipx --version** | Verifica a versão do pipx instalado. |
| **pipx install <pacote>** | Instala um pacote como global no sistema. |
| **pipx list** | Mostra os pacotes instalados como global. |

**POETRY**

Poetry é uma ferramenta para gerenciamento de dependências e empacotamento em Python. Ele permite declarar as bibliotecas que um projeto depende e faz sua gestão (instalar/atualizar) usando apenas um arquivo chamado pyproject.toml.

O que o poetry faz...

* Instalação de bibliotecas.
* Gerenciamento de versões de bibliotecas.
* Gerenciador de ambiente virtual.
* Instalação e distribuição de pacotes.
* Gestão de ambiente de desenvolvimento e testes.

**LINK**

<https://python-poetry.org/>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

pipx install poetry

Fonte: <https://python-poetry.org/docs/#installing-with-pipx>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **poetry** | Lista todas as opções disponíveis para o comando poetry (help) |
| **poetry --version** | Verifica a versão do poetry instalado. |
| **poetry new <nome do projeto>** | Cria um novo projeto no caminho especificado (pasta nova). |
| **poetry init** | Cria um arquivo pyproject.toml básico no diretório atual. |
| **poetry install** | Instala as dependências do projeto baseadas no arquivo pyproject.toml. |
| **poetry add <pacote>** | Adiciona um novo pacote e faz a gestão de suas dependências. |
| **poetry remove <pacote>** | Remove um pacote e suas dependências sem gerar quebra de outro pacote instalado que possui a mesma dependência. |
| **poetry show** | Mostra as dependências instaladas. |
| **poetry shell** | Cria e loga em um shell dentro do ambiente virtual. |
| **poetry env use <versao\_python>** | Define a versão do Python a ser usada pelo ambiente virtual do projeto. Ex. **poetry env use 3.12.1** |

**GIT**

O Git é um sistema de controle de versão distribuído usado para gerenciar projetos de software, mas com aplicações em diversas áreas. Ele oferece um conjunto de ferramentas para o desenvolvimento colaborativo e o rastreamento de alterações em arquivos ao longo do tempo.

O que o git faz...

* Sistema de colaboração.
* Rastreamento de alterações.
* Controle de versão distribuído.

**LINK**

<https://git-scm.com/>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

winget install --id Git.Git -e --source winget

Fonte: <https://git-scm.com/download/win>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **git** | Lista todas as opções disponíveis para o comando git (help) |
| **git --version** | Verifica a versão do git instalada. |
| **git init** | Inicializa um novo repositório. |
| **git clone <url>** | Faz uma cópia (clone) de um repositório existente. |
| **git branch** | Lista todas as branchs no repositório. |
| **git branch <nome da branch>** | Cria uma Branch. |
| **git checkout <nome da branch>** | Mudar de uma branch para outra. |
| **git status** | Mostra o status do diretório |
| **git commit -m “mensagem de commit”** | Realiza um commit com a mensagem descrita. |
| **git log** | Mostra o histórico dos commits. |
| **git push** | Envia as atualizações (commits) para o repositório remoto. |
| **git pull** | Busca as alterações do repositório remoto e integra (atualiza) com a branch local. |

**GITHUB**

O GitHub é uma plataforma de hospedagem de código-fonte baseada na web, que utiliza o sistema de controle de versão Git. É utilizado por desenvolvedores para hospedar, revisar e colaborar em projetos de software, além de oferecer ferramentas para gerenciamento de projetos e integração contínua.

O que o github faz...

* Hospedagem de repositórios Git.
* Colaboração e revisão de código.
* Gerenciamento de problemas (issues) e projetos.

**LINK**

<https://github.com/>

**COMO CRIAR UMA CONTA:**

1. Abrir o site: https://github.com/
2. Clicar no botão Sign Up e preencher os dados solicitados.

Fonte: <https://github.com/>

**PRINCIPAIS COMANDOS**

N/A

**GIT IGNORE (.gitignore) - IGNR**

O arquivo ".gitignore" é um arquivo de configuração usado pelo Git para especificar quais arquivos e diretórios devem ser ignorados durante o controle de versão.

O ignr.py (ou simplesmente ignr) é um utilitário de linha de comando baseado em Python utilizado para gerar o arquivo .gitignore baseado em modelos pré-existentes.

O que o ignr faz...

* Cria arquvos .gitignore com base em modelos pré-existentes.

**LINK**

<https://github.com/Antrikshy/ignr.py>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

pipx install ignr

Fonte: <https://github.com/Antrikshy/ignr.py>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **ignr -h** | Lista todas as opções disponíveis para o comando ignt (help) |
| **ignr -l** | Lista todos os templates disponíveis para o .gitignore no gitignore.io. |
| **ignr -p python** | Gera em tela o conteúdo do arquivo .gitignore para a linguagem python baseado no gitignore.io. |
| **ignr -n python** | Cria um arquivo .gitignore para a linguagem python baseado no gitignore.io no diretório atual. |

**ESTRUTURA DE PASTAS E ARQUIVOS PARA PROJETOS PYTHON**

Uma estrutura de pastas para projetos Python geralmente segue convenções adotadas pela comunidade Python para organizar o código-fonte, testes, documentação de uma maneira padronizada.

O que o estrutura de pastas padronizada faz...

* Organização Lógica do Código.
* Separação de Responsabilidades.
* Documentação Centralizada.
* Facilita a Colaboração.

**COMO CRIAR (SUGESTÃO):**

meu-projeto/ (com traço “–“ semântica, não precisar ter que pressionar shift para informar

│ o nome do projeto)

│

├── README.md

├── LICENSE

├── .python-version

├── .gitignore

├── pyproject.toml

├── requirements.txt

├── requirements-dev.txt

├── (demais arquivos core do projeto)

│

├── logs/

│ └── (logs da aplicação ou projeto – podem ser separados por subpastas por assunto ou escopo)

│

├── docs/

│ └── (documentação do projeto – podem ser separados por subpastas por assunto ou escopo)

│

├── .venv/

│ └── (ambiente virtual do projeto)

│

├── meu\_projeto/ (com sublinhado – padrão pacotes python)

│ ├── \_\_init\_\_.py

│ ├── main.py

│ ├── modulo1.py

│ ├── modulo2.py

│ └── (outros módulos)

│

├── tests/

│ ├── test\_modulo1.py

│ ├── test\_modulo2.py

│ └── (outros testes)

│

├── src/

│ └── (classes, funções, arquivos de código comuns, aplicações – separados por subpastas)

│

├── examples/

│ └── (instruções com códigos de exemplos de uso – separados por subpastas)

│

├── scripts/

│ └── (scripts utilizados nos projetos – separados por subpastas)

│

├── data/

│ └── (arquivos de dados, parquet, csv, json, planilhas – separados por subpastas)

│

├── .github/

│ └── workflows

│ └── (arquivos de CI / CD para o GitHub)

│

├── iac/

│ └── (arquivos de infraestrutura por código com o Terraform – separados por subpastas por recurso cloud)

│

└── assets/

└── (arquivos de imagem, logo, diagramas – separados por subpastas)

**ARQUIVOS COMUNS**

|  |  |
| --- | --- |
| **.gitignore** | Arquivo que contém uma lista de arquivos e pastas que o Git ignora ao fazer commit. |
| **README.md** | Arquivo tipo markdown que fornece uma visão inicial e abrangente do projeto, além de fornecer instruções claras para outros desenvolvedores executarem o projeto com sucesso. |
| **.python-version** | Arquivo que especifica a versão do Python utilizada no projeto, normalmente gerado pelo Pyenv. |
| **.github/workflows** | Contém os scripts de workflow para GitHub Actions, permitindo a automação de pipelines CI/CD diretamente no GitHub. |
| **pyproject.toml** | Arquivo de configuração para ferramentas de construção de projetos Python, como Poetry ou PEP 517/518, contendo metadados do projeto e suas dependências. |
| **requirements.txt** | Uma lista de dependências do projeto para instalação com pip (obsoleto), o pyproject.toml atualmente é o mais comum. |

**SUGESTÃO PARA CRIAÇÃO DO ARQUIVO README**

* **Nome do Projeto**
* **Descrição**
* **Disclaimer ou alertas/avisos**
* **Pré-requisitos:** Informações dos requisitos para usar o projeto (versão do Python, bibliotecas, dependências etc.)
* **Instalação:** Um roteiro com o passo-a-passo para reprodução da configuração do ambiente de dev.
* **Howto:** Exemplos de como utilizar o projeto, com imagens, códigos, etc.
* **Testes:** Como executar os testes criados no seu projeto.
* **Documentação:** Link para a documentação completa do seu projeto.

**PYTEST**

O Pytest é um framework de teste em Python utilizado para escrever e executar testes. Ele simplifica o processo de teste de software, fornecendo uma sintaxe limpa e poderosa, além de uma ampla gama de recursos para facilitar o desenvolvimento e a execução de testes.

O que o pytest faz...

* Testes unitários.
* Testes integrados.
* Testes funcionais.

**LINK**

<https://docs.pytest.org/en/8.2.x/contents.html>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

poetry add --dev pytest

Fonte: <https://docs.pytest.org/en/8.2.x/getting-started.html#install-pytest>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **pytest --version** | Mostra a versão atual do Pytest. |
| **pytest -h** | Lista todas as opções disponíveis para o comando pytest (help) |
| **pytest** | Executa todos os testes no diretório atual. |
| **pytest <nome do arquivo>** | Execute os testes do arquivo informado. |
| **pytest -v** | Habilita a saída com mais detalhes sobre os testes executados. |
| **pytext -vv** | Mostra mais detalhes que o -v. |
| **pytest --html=report.html** | Gera um relatório de testes em formato HTML (precisa do plugin pytest-html). |

**BIBLIOTECAS PARA PADRÃO DE CÓDIGO**

|  |  |  |
| --- | --- | --- |
| **pydocstyle** | Valida a conformidade das docstrings com a convenção PEP257. | <https://www.pydocstyle.org/en/stable/> |
| **isort** | Organiza os imports de bibliotecas de forma ordenada e agrupada. | <https://pycqa.github.io/isort/> |
| **black** | Um formatador de código Python que valida e altera o código em conformidade a PEP 8. | <https://black.readthedocs.io/en/stable/> |
| **blue** | Uma outra opção para formatação de código Python que valida e altera o código em conformidade a PEP 8. | <https://blue.readthedocs.io/en/latest/> |
| **pip-audit** | Busca e corrige vulnerabilidades conhecidas em dependências instaladas. | <https://github.com/pypa/pip-audit> |
| **pylint** |  |  |

**TASKIPY**

O Taskipy é uma ferramenta de automação de tarefas em Python, que permite definir e executar tarefas repetitivas ou complexas por meio de scripts Python.

O que o taskipy faz...

* Cria uma espécie de alias para comandos completos e repetitivos.
* Testes integrados.
* Testes funcionais.

**LINK**

<https://github.com/taskipy/taskipy>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

poetry add --dev taskipy

Fonte: <https://github.com/taskipy/taskipy?tab=readme-ov-file#installation>

**COMO USAR COM POETRY**

Como exemplo de uso, você pode alterar o arquivo **pyproject.toml** e incluir o bloco abaixo:

[tool.taskipy.tasks]

lint = "blue --check --diff . && blue --check --diff ."

test = "pytest --cov=nome\_do\_projeto -vv"

Ele cria dos “atalhos” para as linhas de comandos, simplificando a chamada para:

task lint #(é o mesmo que executar blue --check --diff . && blue --check --diff .)

task test #(é o mesmo que executar pytest --cov=nome\_do\_projeto -vv)

**MKDOCS**

O MKDocs é uma ferramenta utilizada para criar documentação estática a partir de arquivos Markdown.

O que o mkdocs faz...

* Geração de Documentação Estática.
* Simplicidade na Escrita (Markdown).
* Tema Personalizável.
* Integração com Git e GitHub.
* Permite o uso de extensões/plugins (CSS, diagramas etc).

**LINK**

<https://www.mkdocs.org/>

**COMO INSTALAR (Windows):**

1. Abrir o Powershell no Windows.
2. Executar o comando:

poetry add mkdocs

Fonte: <https://www.mkdocs.org/getting-started/>

**PRINCIPAIS COMANDOS**

|  |  |
| --- | --- |
| **mkdocs --version** | Mostra a versão atual do mkdocs. |
| **mkdocs** | Lista todas as opções disponíveis para o comando mkdocs (help) |
| **mkdocs new <nome do projeto>** | Cria o arquivo mkdocs.yaml e uma pasta /docs com o arquivo index.md. |
| **mkdocs serve** | Visualiza localmente no formato web as páginas em construção. |
| **mkdocs build** | Constrói o site de documentação. Cria uma pasta chamada /site com os arquivos no formato web (html, css, js etc.) |
| **mkdocs gh-deploy** | Publica a documentação gerada pelo build no GitHub Pages (<https://usuario.github.io/repositorio/>). |