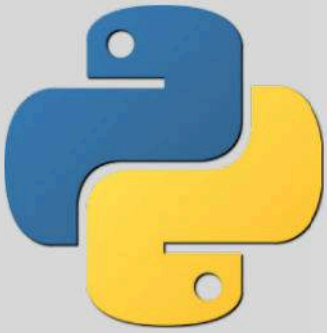
Python Fundamentals



Módulos e Instaladores de Pacotes

7.1

Anotações

Módulos e Instaladores de Pacotes

Objetivos

- ✔ Compreender a criação de módulos.
- ✔ Trabalhar com instalação utilizando o easy_install.
- ✔ Trabalhar com instalação utilizando o pip.

Instaladores de Pacotes

Nesta aula abordaremos o conceito de módulos, como cria-los, sua instalação via easy_install e pip.

Anotações

Módulos e Instaladores de Pacotes

✔ Um módulo, nada mais é que um arquivo com várias funções utilizáveis em outra parte do código.

✔ No Python utilizamos alguns módulos já prontos ou criamos os nossos próprios módulos.

✔ É extremamente importante modularizar aplicações, desta forma reutilizamos os módulos em várias partes do projeto, facilitando manter e controlar.

Anotações

Criando um módulo

Crie um arquivo, com o nome module.py, defina duas funções:

```
#!/usr/bin/python3
def mod():
    print("Modulo 1")

def mod2():
    print("Modulo 2")
```

Anotações

Utilizando o módulo

Como usá-lo na aplicação. O primeiro passo, importar o módulo:

```
>>> import module
>>> type (module)
<class 'module'>
```

Anotações

Utilizando o módulo

Veja informações sobre o módulo:

```
>>> dir(module)
[ … 'mod', 'mod2']
```

Chame o módulo:

```
>>> module.mod()
Modulo 1
```

Anotações

Utilizando o módulo

Para importar um módulo específico, execute:

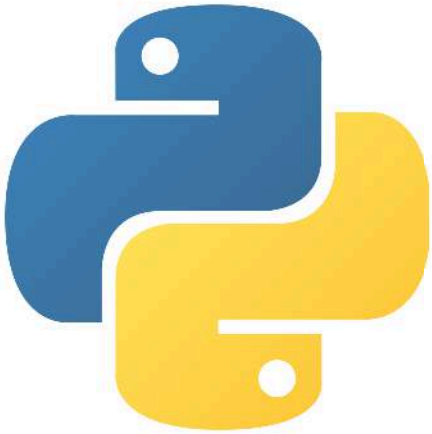
```
>>> from module import mod
>>> mod()
Modulo 1
```

Anotações

Instalando Módulos Externos

Python dispõem de um repositório oficial chamado PyPI (Python Package Index), onde são disponibilizados módulos para uso em aplicação.

`https://pypi.python.org/pypi`



O PyPI (Python Package Index) possui uma lista enorme de pacotes utilizáveis. Atualmente existem mais de 100 mil pacotes disponíveis.

Existem módulos para fazer integração com diversas ferramentas do dia a dia como:

Jenkins: https://pypi.python.org/pypi/python-jenkins/0.4.12

GitLab: https://pypi.python.org/pypi/pyapi-gitlab/7.8.5

MongoDB: https://pypi.python.org/pypi/pymongo/3.2

Há módulos para trabalhar com testes:

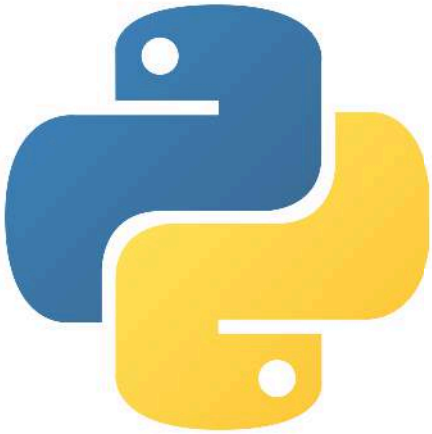
Teste Funcional
Selenium: https://pypi.python.org/pypi/selenium

Teste Unitário
Pytest: https://pypi.python.org/pypi/pytest

Instalando Módulos Externos

Utilize o pacote python-setuptools que disponibiliza o comando easy_install usado para a instalação dos módulos .

```
easy_install pip3
easy_install -U pip3
```



O comando **easy_install** origina-se do pacote setuptools, desenvolvido em 2004 como a primeira ferramenta para instalação de pacotes do python. Logo destacou-se, devido a facilidade em conectar no repositório PyPI, resolvendo as dependências.

Anteriormente, a instalação acontecia baixando o código fonte do pacote e, utilizando os seguintes comandos para efetuar a instalação: python setup.py install

Anotações

Instalando Módulos Externos

Outra maneira de instalar: use o pacote python3-pip que disponibiliza o comando pip.

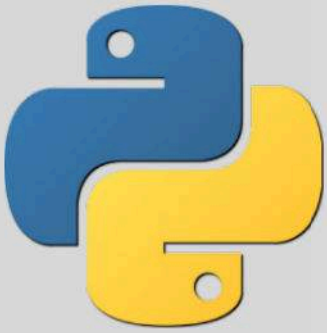
```
pip install modulo = instala um módulo
pip search modulo = faz a busca pelo nome do módulo
pip list = lista todos os módulos instalados
pip uninstall = exclui um módulo
```

Como o comando **pip** foi baseado no pacote setuptools, tolera ser instalado através do comando easy_install.

pip foi desenvolvido em 2008 como alternativa ao easy_install, tornou-se bem popular devido a facilidade de uso. Além de possuir mais opções que o comando easy_install, é mais rápido, não realiza a instalação de um pacote a partir do zero, provê apenas os metadados com códigos fonte essenciais para o funcionamento do módulo.

Anotações

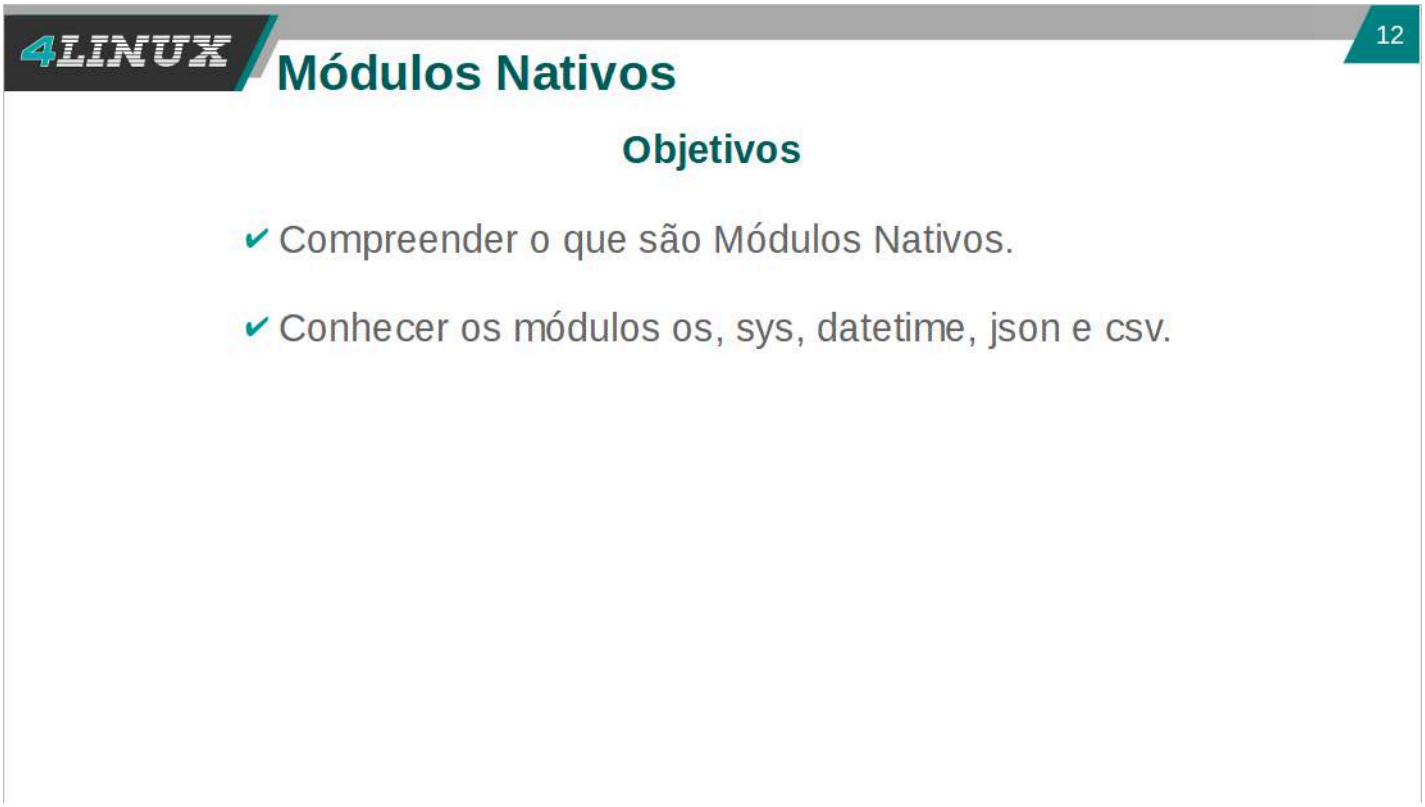
Python Fundamentals



Módulos Nativos

7.2

Anotações



Módulos Nativos

Nesta aula apresentaremos módulos Nativos, você aprenderá sobre suas funcionalidades e porque utilizá-los em aplicações.

Anotações

Sintaxe

- ✔ Em Python, criamos novos módulos utilizando módulos externos (prontos) ou empregamos módulos nativos.
- ✔ Os módulos nativos são aqueles embutidos na linguagem, não é necessário instalar, basta realizar o “import” e utilizá-los conforme a sua sintaxe.

Anotações

Módulos Nativos do Python

Os: possibilita usar funcionalidades do sistema operacional.

Sys: permite acessar parâmetros e funções específicas.

Datetime: traz alguns tipos de data e hora.

Json: codifica e decodifica no formato JSON.

Csv: lê e escreve arquivos no forrmato CSV (planilhas).

Anotações

Módulo “os”

- ✔ Utilize sempre ao criar um script que precise executar algum comando no linux.
- ✔ Exemplo: criar diretórios, ver informações do sistema, listar arquivos, ver informações de usuário, entre outras coisas.

Anotações

Funcionalidades Módulo “os”

```
import os
#ver qual o usuário logado
print(os.getlogin())
#listar o conteúdo de um diretório
print(os.listdir('/caminho/do/diretorio'))
#renomear um arquivo
print(os.rename('nome_atual','novo_nome'))
#executar qualquer comando
os.system('digite o comando aqui')
```

Modulo OS

Utiliza-se este módulo quando se executa algo no sistema operacional Linux. Consiste o módulo Python que fará chamadas de sistema, executando o comando.

Utilizando “os.system('comando’), executamos o módulo. Porém, recomenda-se consultar na documentação a melhor forma para a execução, pode existir um modo específico para realizar o que planeja.

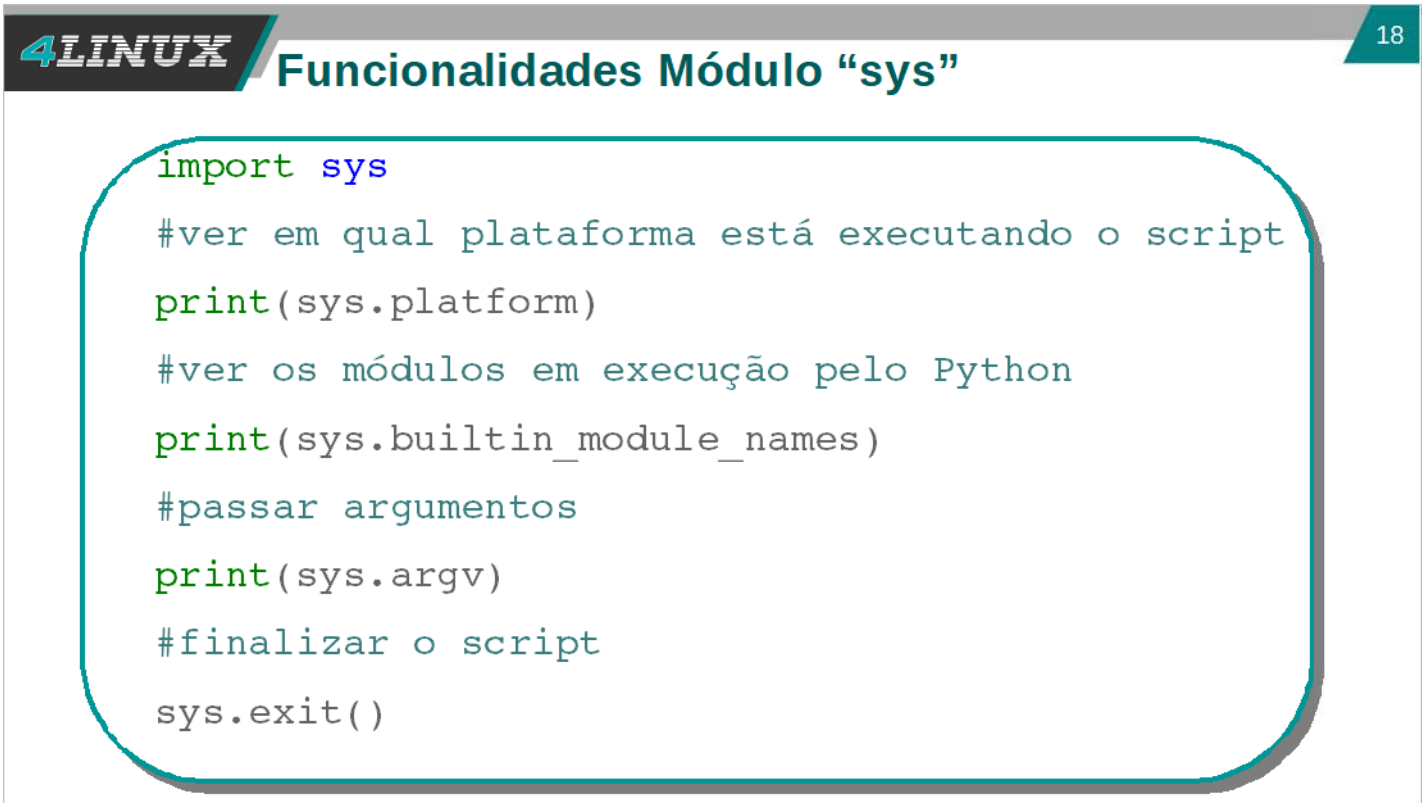
Para maiores informações sobre o módulo os, verifique a documentação oficial: https://docs.python.org/3/library/os.html#module-os

Anotações

Módulo “sys”

- ✔ Fornece acesso a algumas variáveis ou funções executadas pelo Python.
- ✔ Permite trabalhar com o interpretador, conseguimos ver a versão do Python, em que sistema está em execução, passar parâmetros, entre outras coisas.

Anotações



Modulo Sys

Traz informações sobre o interpretador (python). Funcionalidade muito interessante, permite passar argumentos, veja o exemplo:

```
#!/usr/bin/python3
import sys

for i in range(len(sys.argv)):
    if i == 0:
        print("Function name: {}".format(sys.argv[0]))
    else:
        print("{}. argument: {}" .format(i,sys.argv[i]))
```

argv retornará os argumentos como uma lista, ao passar 3 argumentos, para acessá-los: sys.argv[1], sys.argv[2] e sys.argv[3]. sys.argv[0], corresponde ao nome do Script.

Para conhecer mais sobre o módulo, veja a documentação oficial:
https://docs.python.org/3/library/sys.html#module-sys

Módulo “Datetime”

- ✔ Fornece formas de manipular data e hora, de maneiras simples a complexas.
- ✔ É muito importante saber trabalhar com data e hora, por esta razão a linguagem oferece este módulo nativo.

Anotações

Funcionalidades Módulo “Datetime”

```
import datetime
#mostra a data atual
print(datetime.datetime.now())
# mostra a data após 7 dias
print(datetime.timedelta(7))
# definir um horário, por exemplo: 14hrs
print(datetime.time(14,0,0))
# definir uma data, por exemplo: 1 de janeiro de 2017
print(datetime.date(2017,1,1))
```

Modulo Datetime

Permite a manipulação de data e horário. Veja o exemplo, expirar token com validade de 1h:

```
#!/usr/bin/python3
from datetime import datetime
acesso = datetime(2018,01,22,00,00,00)
atual = datetime(2018,01,22,01,01,00)

if (atual - acesso).total_seconds() > 3600:
    print("Seu token expirou")
else:
    print("Acesso liberado")
```

Neste exemplo prático, de cálculo do termino de tempo de um token de acesso, realizamos a subtração de 2 objetos Datetime, a operação retorna um outro objeto Datetime com a diferença de datas. Empregamos o método total_seconds, que converterá o resultado em segundos, comparando com 3600 segundos o equivalente a 1 hora. Se o total de segundos for superior 3600 o token será expirado.

Para saber mais sobre o módulo datetime, veja a documentação oficial:
https://docs.python.org/3/library/datetime.html#module-datetime

Módulo “JSON”

- ✔ Permite mudar determinados caracteres para o formato JSON e vice-versa.
- ✔ Módulo muito importante. Ao trabalhar com API’s, normalmente usamos este formato, sendo necessário realizar a conversão para utilizar os dados na aplicação.

Anotações

Funcionalidades Módulo “JSON”

```
import json
#transformar o json em string
print(json.dumps({"nome":"python",
                  "modulo":"fundamentals"})
#transformr a string em json
print(json.loads('{"nome":"python",
                   "modulo":"fundamentals"}'))
```

Modulo Json

A integração com outras ferramentas, normalmente requer uso de APIs para envio e recebimento De dados. É comum tais dados virem no formado JSON. Como um dicionário, as vezes a aplicação não combina com este tipo de formato, sendo necessário converter este JSON para string e vice-versa. No código apresentado, conferimos cada tipo.

```
#!/usr/bin/python3
import json
print(type(json.dumps({"nome":"python",
                       "modulo":"fundamentals"})))

print(type(json.loads('{"nome":"python",
                        "modulo":"fundamentals"}')))
```

Estude a documentação oficial para conhecer mais sobre o módulo json: https://docs.python.org/3/library/json.html#module-json

Módulo “CSV”

- ✔ Viabiliza que um script trabalhe com planilhas, fazendo importação e exportação.
- ✔
- ✔ Permite ler e escrever arquivos deste tipo, usando os dados em nossos scripts.

Anotações

Ver informações em arquivo

```
import csv

with open('teste.csv', 'r') as csvfile:
    arquivo = csv.reader(csvfile, delimiter=' ')
    for a in arquivo:
        print(a)
```

Modulo csv

Assemelha-se a abertura de arquivos txt e binários do python. Porém, dispõe de parâmetros que facilitam o trabalho com este tipo de formato. Por exemplo o parâmetro 'delimiter', na leitura, determina qual separação de dados ocorrerá por virgulas, espaços, pipes etc…

Anotações

Gravar informações em arquivo

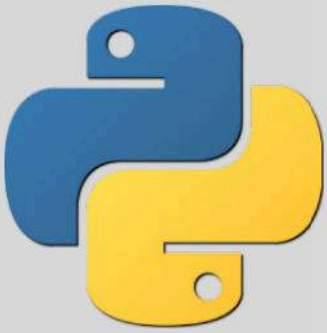
```
import csv

with open('arquivo.csv', 'w', newline='') as csvfile:
    arquivo = csv.writer(csvfile, delimiter=',')
    arquivo.writerow(['Teste'] * 5)
    arquivo.writerow(['4linux',
                      'Python'])
```

Modulo csv

Possibilita realizar métodos de escrita avançados, facilitando a inserção de dados formatados em massa que serão lidos por algum software (leitor de planilhas).

Anotações

Python Fundamentals



Ambientes Isolados

7.3

Anotações

Ambientes isolados

Objetivos

- ✔ Compreender a utilidade de ambiente isolado.
- ✔ Conhecer o módulo virtualenv.

Ambiente isolado

Você compreenderá o que é ambiente isolado e a vantagem ao utilizar o modulo virutalenv.

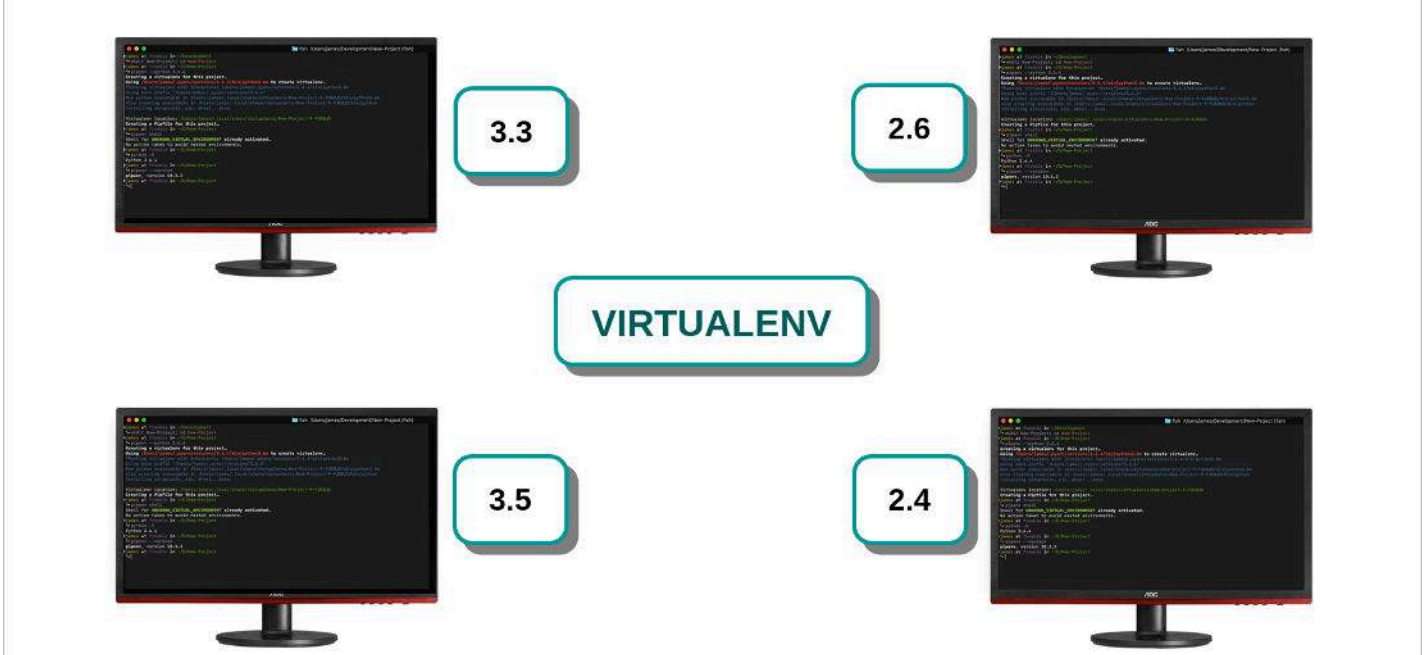
Anotações

Virtual Environment

- ✔ Dispositivo que permite criar, ambientes virtuais isolados para projetos Python.
- ✔ Por exemplo: possibilita criar ambientes virtuais para dois projetos, A e B, um ambiente para cada projeto.
- ✔ Ao criar os ambientes, por exemplo: venvA e venvB, serão gerados diretórios com o nome de cada ambiente.

Anotações

Virtualenv



Anotações

Virtual Environment

- ✔ Viabiliza manter as duas versões do mesmo módulo sem causar conflito.
- ✔ Exemplo: uma aplicação requer a versão 1 do modulo (X), entretanto, outra aplicação usa o mesmo módulo da versão 2.

Anotações

Instalação e criação do ambiente

1	Instale o Virtualenv através do gerenciador de pacotes pip: # pip3 install virtualenv
2	Para criar o virtualenv, basta informar o interpretador default e o nome do virtualenv: # virtualenv -p /usr/bin/python3 venv
3	Para ativar as variáveis do projeto criado, execute: # source venv/bin/activate
4	Para desativar as variáveis e voltar a utilizar o bash padrão, execute: # deactivate

Virtualenv

Constitui boa prática manter uma virtualenv para cada projeto com seus respectivos módulos, evitando conflito de versionamento.

Anotações

Instalação e criação do ambiente

1 Para visualizar os módulos instalados:

```
# pip3 freeze
```

2 Direcionar todos os módulos do projeto para um arquivo txt:

```
# pip3 freeze > requeriments.txt
```

3 Instalar todos os módulos escritos no arquivo txt.

```
# pip3 install -r requeriments.txt
```

Virtualenv

Consiste uma boa prática, manter em cada projeto um arquivo txt com seus respectivos módulos, assegurando que plataformas, até mesmo outros colaboradores do projeto, consigam instalar todos os módulos necessários para rodar a aplicação.

Anotações