SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, evoluiremos nosso conhecimento sobre a criação de módulos e bibliotecas em Python. Vamos explorar a importância dessa prática para a organização e a reutilização de código, bem como aprender a criar nossos próprios módulos e bibliotecas de maneira eficiente.

HANDS ON

Nesta aula prática, demonstraremos como criar e publicar uma biblioteca no PyPI. Exploramos a importância dessa prática para a organização e a reutilização de código, além de aprendermos a criar nossos próprios módulos e bibliotecas de maneira eficiente.

SAIBA MAIS

A criação de módulos e bibliotecas é um passo fundamental para a organização e a reutilização do código em Python. Nesta aula, exploraremos como criar módulos e bibliotecas eficientes, destacando a importância dessas práticas e oferecendo um passo a passo detalhado.

Mas antes de avançarmos para a prática, vejamos alguns pontos teóricos, como o que são módulos e bibliotecas. Iniciando pelos **módulos,** eles são arquivos Python contendo definições de funções, classes e variáveis e permitem a organização do código em blocos reutilizáveis.

Já as **bibliotecas** são coleções de módulos que oferecem funcionalidades específicas. Elas são essenciais para a criação de softwares robustos e para facilitar o trabalho com determinadas tarefas ou áreas de aplicação.

Agora, acredito que você tenha o seguinte questionamento: por que devemos criar módulos e bibliotecas? Para responder esta e outras perguntas sobre o tema, separamos alguns tópicos:

- **Organização**: facilita a manutenção e a leitura do código.
- **Reutilização**: permite que códigos já desenvolvidos sejam reutilizados em diferentes projetos.
- **Facilidade de testes**: isola funcionalidades em módulos, facilitando a criação de testes unitários.
- **Escalabilidade**: torna o desenvolvimento de projetos maiores mais gerenciáveis.

Agora que sabemos o que são módulos e bibliotecas, vamos criar um módulo com funções para calcular retornos de investimento, crescimento composto e conversão de taxas de juros.

Para isso, crie um arquivo chamado `investimentos.py` e adicione o seguinte código:

```python
# investimentos.py

def calcular_retorno_investimento(valor_inicial, valor_final):
    """
    Calcula o retorno de investimento.

    Args:
        valor_inicial (float): Valor inicial do investimento.
        valor_final (float): Valor final do investimento.

    Returns:
        float: Retorno do investimento em porcentagem.
    """
    retorno = (valor_final - valor_inicial) / valor_inicial * 100
    return retorno

def calcular_juros_compostos(principal, taxa_juros_anual, periodos):
    """
    Calcula o valor final de um investimento com juros compostos.

    Args:
        principal (float): Valor inicial investido.
        taxa_juros_anual (float): Taxa de juros anual em porcentagem.
        periodos (int): Número de períodos (anos).

    Returns:
        float: Valor final após o período com juros compostos.
    """
    taxa_juros_decimal = taxa_juros_anual / 100
    valor_final = principal * (1 + taxa_juros_decimal) ** periodos
    return valor_final

def converter_taxa_anual_para_mensal(taxa_anual):
```

```
    """
    Converte uma taxa de juros anual para mensal.

    Args:
        taxa_anual (float): Taxa de juros anual em porcentagem.

    Returns:
        float: Taxa de juros mensal em porcentagem.
    """
    taxa_mensal = (1 + taxa_anual / 100) ** (1 / 12) - 1
    return taxa_mensal * 100

def calcular_cagr(valor_inicial, valor_final, anos):
    """
    Calcula a taxa de crescimento anual composta (CAGR).

    Args:
        valor_inicial (float): Valor inicial do investimento.
        valor_final (float): Valor final do investimento.
        anos (int): Número de anos.

    Returns:
        float: CAGR em porcentagem.
    """
    cagr = ((valor_final / valor_inicial) ** (1 / anos) - 1) * 100
    return cagr
```

Agora, crie outro arquivo chamado main.py e atualize ele com o seguinte código para importar seu módulo:

```
# main.py
```

```python
import investimentos

valor_inicial = 1000
valor_final = 1500
anos = 5
taxa_anual = 6

retorno = investimentos.calcular_retorno_investimento(valor_inicial, valor_final)
print(f"Retorno do investimento: {retorno:.2f}%")

valor_final_juros = investimentos.calcular_juros_compostos(valor_inicial,
taxa_anual, anos)
print(f"Valor final com juros compostos: R${valor_final_juros:.2f}")

taxa_mensal = investimentos.converter_taxa_anual_para_mensal(taxa_anual)
print(f"Taxa de juros mensal: {taxa_mensal:.2f}%")

cagr = investimentos.calcular_cagr(valor_inicial, valor_final, anos)
print(f"CAGR: {cagr:.2f}%")
```

Resultado:

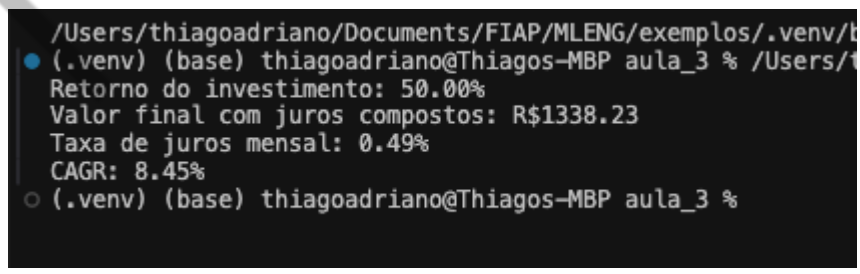

Figura 1: Resultado da execução do módulo de investimentos
Fonte: Elaborado pelo autor (2024)

Agora, para criar uma biblioteca, você precisará organizar vários módulos em um diretório e criar um arquivo `setup.py` para definir suas informações. Para ilustrar isto, vamos conceber uma nova biblioteca com o módulo para cálculo de investimentos.

Para isso, crie a seguinte estrutura de pastas e arquivos:

```
meu_investimento/
│
├── investimentos/
│   ├── __init__.py
│   └── investimentos.py
│
├── tests/
│   └── test_investimentos.py
│
├── setup.py
├── README.md
└── LICENSE
```

Agora, vamos popular cada um deles. Iniciando pelo arquivo `__init__.py` dentro da pasta `investimentos`, é possível que o diretório seja tratado como um pacote. Ele pode estar vazio ou importar funções do módulo `investimentos.py`.

```
from .investimentos import calcular_retorno_investimento,
calcular_juros_compostos, converter_taxa_anual_para_mensal, calcular_cagr
```

Agora, atualize o setup.py: este é o script de configuração usado pelo setuptools para criar o pacote.

Aqui está um exemplo de como ele pode ser configurado:

```
from setuptools import setup, find_packages

setup(
    name='meu_investimento',
    version='0.1',
    packages=find_packages(),
    install_requires=[],
```

```
    author='Thiago S Adriano',
    author_email='tadriano.dev@gmail.com',
    description='Uma biblioteca para cálculos de investimentos.',
    url='https://github.com/tadrianonet/meu_investimento',
    classifiers=[
        'Programming Language :: Python :: 3',
        'License :: OSI Approved :: MIT License',
        'Operating System :: OS Independent',
    ],
    python_requires='>=3.6',
)
```

README.md

```
# Meu Investimento

Uma biblioteca Python para cálculos de investimentos.

## Instalação

Você pode instalar a biblioteca via pip:

```bash
pip install meu_investimento
```

Uso

```python
from investimentos import calcular_retorno_investimento,
calcular_juros_compostos
```
```

```
valor_inicial = 1000
valor_final = 1500

retorno = calcular_retorno_investimento(valor_inicial, valor_final)
print(f"Retorno do investimento: {retorno:.2f}%")

valor_final_juros = calcular_juros_compostos(valor_inicial, 6, 5)
print(f"Valor final com juros compostos: R${valor_final_juros:.2f}")
```

Agora, para verificar se tudo está ok, é sempre importante criarmos os nossos testes: tests/test_investimentos.py.

```
# tests/test_investimentos.py

import unittest
from investimentos import calcular_retorno_investimento,
calcular_juros_compostos, converter_taxa_anual_para_mensal, calcular_cagr

class TestInvestimentos(unittest.TestCase):

    def test_calcular_retorno_investimento(self):
        self.assertAlmostEqual(calcular_retorno_investimento(1000, 1500), 50.0)

    def test_calcular_juros_compostos(self):
        self.assertAlmostEqual(calcular_juros_compostos(1000, 6, 5), 1338.23,
places=2)

    def test_converter_taxa_anual_para_mensal(self):
        self.assertAlmostEqual(converter_taxa_anual_para_mensal(12), 0.9487,
places=3)  # Alterado de places=4 para places=3
```

```
  def test_calcular_cagr(self):
      self.assertAlmostEqual(calcular_cagr(1000, 1500, 5), 8.45, places=2)

if __name__ == '__main__':
  unittest.main()
```

Para disponibilizar sua biblioteca para a comunidade Python, você pode publicá-la no PyPI (Python Package Index).

E para te ajudar nesta etapa, é só reassistir nossa videoaula, em que demonstraremos esse passo a passo para publicar uma biblioteca no PyPI.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, evoluímos nosso conhecimento sobre a criação de módulos e bibliotecas em Python. Exploramos a importância dessa prática para a organização e a reutilização de código, bem como aprendemos a criar nossos próprios módulos e bibliotecas de maneira eficiente.

REFERÊNCIAS

MARTINS, V. **Construindo uma Biblioteca Python do Zero — Parte 1**. 2023. Disponível em: <https://medium.com/@vinicius.pereira.mts/construindo-uma-biblioteca-python-do-zero-parte-1-4fda732fa0f0>. Acesso em: 15 ago. 2024.

PYTHON.ORG. **6. Módulos**. 2024. Disponível em: <https://docs.python.org/pt-br/3/tutorial/modules.html>. Acesso em: 15 ago. 2024.

STURTZ, J. **Python Modules and Packages**: An Introduction. realpython. 2023. Disponível em: <https://realpython.com/python-modules-packages/>. Acesso em: 15 ago. 2024.

PALAVRAS-CHAVE

Palavras-chave: Python. Module. Library.