Backend
Python com Django

Prof. Bruno Iran Ferreira Maciel

Aug 2024

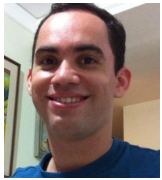
Estrutura de Tópicos das Aulas

Resumo do Conteúdo Programático

N	Aulas	Mês	Data	Conteúdo Previsto
1	1,5	Agosto	02/08/2024	Apresentação e boas-vindas
2	3	Agosto	02/08/2024	Lógica de Prog. e Padrões de Des. de Software
3	4,5	Agosto	03/08/2024	Lógica de Programação
4	6	Agosto	03/08/2024	Padrões de Desenvolvimento de Software
5	7,5	Agosto	09/08/2024	Lógica de Programação
6	9	Agosto	09/08/2024	Padrões de Desenvolvimento de Software
7	10,5	Agosto	10/08/2024	POO
8	12	Agosto	10/08/2024	Padrões de Desenvolvimento de Software
9	13,5	Agosto	16/08/2024	POO
10	15	Agosto	16/08/2024	Git
11	16,5	Agosto	17/08/2024	POO
12	18	Agosto	17/08/2024	Padrões de Desenvolvimento de Software
13	19,5	Agosto	23/08/2024	POO
14	21	Agosto	23/08/2024	Padrões de Desenvolvimento de Software
15	22,5	Agosto	24/08/2024	POO
16	24	Agosto	24/08/2024	Padrões de Desenvolvimento de Software
17	25,5	Agosto	30/08/2024	POO
18	27	Agosto	30/08/2024	Padrões de Desenvolvimento de Software
19	28,5	Agosto	31/08/2024	POO
20	30	Agosto	31/08/2024	Soft Skills
21	31,5	Setembro	06/09/2024	POO
22	33	Setembro	06/09/2024	Padrões de Desenvolvimento de Software
23	0	Setembro	07/09/2024	Feriado Nacional - Independência do Brasil
x		Setembro	13/09/2024	POO
x		Setembro	13/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	14/09/2024	POO
x		Setembro	14/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	20/09/2024	POO
x		Setembro	20/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	21/09/2024	POO

N	Aulas	Mês	Data	Conteúdo Previsto
x		Outubro	04/10/2024	Django
x		Outubro	05/10/2024	Soft Skills
x		Outubro	05/10/2024	POO
x		Outubro	11/10/2024	Django
x		Outubro	11/10/2024	Web Services
x		Outubro	12/10/2024	Feriado Nacional - Nossa Senhora Aparecida
x		Outubro	18/10/2024	Django
x		Outubro	18/10/2024	Web Services
x		Outubro	19/10/2024	Django
x		Outubro	19/10/2024	Web Services
x		Outubro	25/10/2024	Django
x		Outubro	25/10/2024	Web Services
x		Outubro	26/10/2024	Django
x		Outubro	26/10/2024	Web Services
x		Novembro	01/11/2024	Django
x		Novembro	01/11/2024	Web Services
x		Novembro	02/11/2024	Feriado Nacional - Finados
x		Novembro	08/11/2024	Django
x		Novembro	08/11/2024	Web Services
x		Novembro	09/11/2024	Soft Skills
x		Novembro	09/11/2024	Django
x		Novembro	15/11/2024	Feriado Nacional - Proclamação da República
x		Novembro	16/11/2024	Imprensado - Não teremos aulas
x		Novembro	22/11/2024	Django
x		Novembro	22/11/2024	Web Services
x		Novembro	23/11/2024	Django
x		Novembro	23/11/2024	Web Services
x		Novembro	29/11/2024	Django
x		Novembro	29/11/2024	Web Services
x		Novembro	30/11/2024	Django
x		Novembro	30/11/2024	Web Services

Quem sou eu?



Prof. Dr. Bruno Iran Ferreira Maciel, 41

Atuo como professor; desenvolvedor; e consultor de TI.

- Doutor em Ciência da Computação, 2015-2020
- Mestre em Ciência da Computação, 2012-2014
- Especialista em Engenharia e Reúso de Software, 2011-2012
- Graduado em Sistemas de Informação, 2016-2016
- Graduado em Ciência da Computação, 2007-2011
- Técnico em Análise e Desenvolvimento de Software, 2007-2007
- CV completo http://bit.ly/brunomaciel-lattes

Apresentação pessoal, integração com a turma, introdução de conceitos básicos de desenvolvimento de softwae e despertar curiosidade dos alunos sobre o tema.

Compromisso semanal

Encontros: sextas e sábados

- Período: 02/08/2024 à 30/11/2024

Sextas

- Início: 18h
- Térmico: 21h - Sem intervalo
- Térmico: 21h10 - Com intervalo de 10 minutos (a confirmar)

Sábados

- Início: 9h
- Térmico: 12h - Sem intervalo
- Térmico: 12h10 - Com intervalo de 10 minutos (a confirmar)

Reposições de aulas aos Sábados no período da tarde

- Início: 14h
- Térmico: 17h - Sem intervalo
- Térmico: 17h10 - Com intervalo de 10 minutos (a confirmar)

Metodologia das Aulas

Aulas:

- 18h-19h30
- 19h30-21h

- Resolução de dúvidas gerais e tolerância: 18h até 18h15 (15 minutos)
- Revisão da aula passada: 18h15 até 18h30 (15 minutos)
- Adição de novo conteúdo: 18h30 até 21h (2h30)

EMENTA

O curso tem como objetivo desenvolver as habilidades necessárias em programação, noções de padrões de desenvolvimento de software, arquitetura cliente-servidor, noções de banco de dados e frameworks back-end para que você seja capaz de projetar soluções web que sejam seguras, robustas e escaláveis, com base em tecnologias modernas e nas melhores práticas de desenvolvimento de software.

OBJETIVO

Compreender o funcionamento de características e arranjos básicos de desenvolvimento de software com foco em backend.
Permitir análise crítica das questões relativas aos conceitos estudados ao longo das aulas, bem como a identificação de áreas de pesquisa voltadas para o aperfeiçoamento das técnicas e desenvolvimento de novas aplicações.

Competências Específicas

- Lógica de Progamação.
- Padrões de projetos.
- Soft Skills.
- Padrões de projetos.
- POO.
- Git.
- Web Services.
- Django.

Github das aulas

https://github.com/brunom4ciel/material-python

Bibliografia básica

- Ghetan, Giridhar. Aprendendo Padrões de Projeto em Python, 1ª Edição. Novatec. 2016. ?

Acrônimo 1

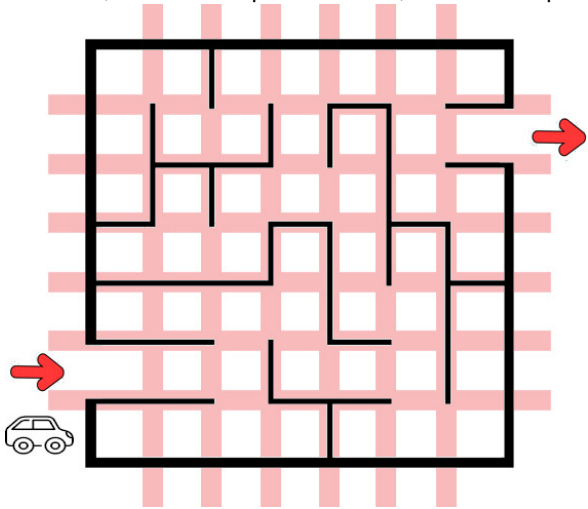
Apresentação e boas-vindas

Apresentação e boas-vindas

Olá?

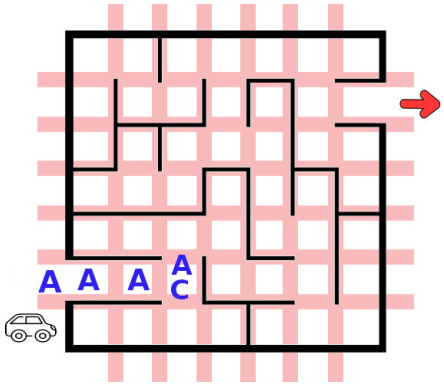
Desafio

* A = Passo a frente, * B = Vire para a direita, * C = Vire para a esquerda



Desafio

A - Passo a frente
A - Passo a frente
A - Passo a frente
C - Vire para a esquerda
A - Passo a frente
B - Vire para a direita
A - Passo a frente
C - Vire para a esquerda
A - Passo a frente
A - Passo a frente



Lógica de Programação com Django

A lógica de programação é a forma como são conduzidas as ações realizadas por algoritmos.

- Toda programação apresenta um encadeamento lógico para que os códigos descritos possam executar os comandos atribuídos.
- Nesse sentido, quem programa é responsável por compreender essa lógica e traduzi-la de forma eficiente para a máquina.

Exercícios

- *Exercício 001*: Ler 4 valores (considere que não serão informados valores iguais). Escreva a soma dos dois últimos números.
- *Exercício 002*: Ler 2 valores e se o segundo valor informado for ZERO, deve ser lido um novo valor, ou seja, para o segundo valor não pode ser aceito o valor zero e imprimir o resultado da divisão do primeiro valor lido pelo segundo valor lido. (utilizar a estrutura REPETIR)
- *Exercício 003*: Ler as idades de 2 homens e de 2 mulheres (considere que as idades dos homens serão sempre diferentes entre si, bem como as das mulheres). Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova, e o produto das idades do homem mais novo com a mulher mais velha.
- *Exercício 004*: Ler o salário fixo e o valor total das vendas efetuadas pelo vendedor de uma empresa. Sabendo-se que ele recebe uma comissão de 3% sobre o total das vendas até R$ 1.500,00 mais 5% sobre o que ultrapassar este valor, calcular e escrever o seu salário total.

Lógica de Prog. e Padrões de Des. de Software

Padrões de desenvolvimento de software

Design patterns, também conhecidos como padrões de design ou padrões de projeto, são soluções já testadas e aprovadas em problemas recorrentes durante o desenvolvimento de software.

- **abstração**: os padrões representam um conhecimento aplicado no cotidiano.
- **encapsulamento**: um problema é retratado como uma cápsula, que deve ser independente, específica e objetiva. O mesmo vale para uma solução já definida.
- **combinatoriedade**: há uma hierarquia entre os padrões, do mais alto ao mais baixo.
- **equilíbrio**: cada passo dentro de um projeto de software precisa buscar o equilíbrio.
- **abertura**: os padrões precisam permitir uma extensão até os níveis mais baixos.
- **generalidade**: todo padrão deve servir de base para a construção de outros projetos.

Lógica de Programação

Lógica de Programação com Django

Tutorial de boas-vindas com Django.

- instalação: faça a instalação do django. Comece repetindo o comando abaixo no terminal e se funcionar já era....

```
|> py -m pip install Django
```

- criação do projeto: entre no diretório que pretende salvar o projeto e repita o comando abaixo no terminal.

```
|> django-admin startproject django_project .
```

- execução: entre no diretório do projeto que foi criado e repita o comando abaixo no terminal.

```
|> python manage.py runserver
```

- criação de app: repita o comando abaixo no terminal.

```
|> python manage.py startapp pages
```

Lógica de Programação com Django

...continuando....procure o arquivo -> pages/views.py

```
from django.http import HttpResponse

def home_page_view(request):
    return HttpResponse("Olá, mundo!")
```

Lógica de Programação com Django

...continuando....procure o arquivo -> pages/urls.py

```
from django.urls import path
from .views import home_page_view

urlpatterns = [
    path('', home_page_view),
]
```

Lógica de Programação com Django

...continuando....procure o arquivo -> django_project/urls.py

```
from django.contrib import admin
from django.urls import path, include # new

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("pages.urls")), # new
]
```

Padrões de Desenvolvimento de Software

Padrões de desenvolvimento de software

Para que servem e quando usar os design patterns?

- O principal objetivo dos padrões de design em desenvolvimento é deixar o código mais fácil de ser mantido e testado.

Padrões de desenvolvimento de software

Quais as vantagens dos padrões de design em desenvolvimento?

- Mesmo que os problemas de um projeto para outro não sejam iguais, há similaridades. Por focar na reutilização de soluções já testadas e aprovadas, os padrões de design oferecem maior agilidade e flexibilidade ao dia a dia dos desenvolvedores.

Padrões de desenvolvimento de software

Se os padrões de design ajudam a resolver problemas recorrentes em um software a partir de um modelo, é importante conhecer os três grupos. São eles: Creational (Criação), Structural (Estrutura) e Behavioral (Comportamental).

- **Criacional**: os padrões de criação priorizam a interface e, por isso, lidam com a criação de objetos.
- **Estrutural**: os padrões estruturais envolvem a relação entre os objetos, ou seja, a forma como eles interagem entre si
- **comportamental**: os padrões comportamentais estão diretamente ligados à comunicação entre os objetos.

Exercícios

- *Exercício 005*: Ler 11 valores numéricos, somar os 10 primeiros e guardar em uma variável A e o décimo primeiro valor, guardar em uma variável B. Escreva os valores de A e B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final, escreva os valores que ficaram armazenados nas variáveis.
- *Exercício 006*: Ler um valor numérico e escrever o seu antecessor. Ex: Ler n = 20, Escreva 19.
- *Exercício 007*: Ler três valores que representam a idade de uma pessoa, expressa em anos, meses e dias (data de nascimento). Escreva a idade dessa pessoa expressa apenas em dias. Considerar ano com 365 dias e mês com 30 dias.
- *Exercício 008*: Ler o número total alunos de uma sala de aula, o número de votos em candidato A e candidato B. Escreva o percentual que cada candidato representa em relação ao total de alunos. Considere que o número total de alunos votou no candidato A ou B.

Exercícios

- *Exercício 009*: Sistema de ordenação de valores. Ler 5 valores (considere que não serão informados valores iguais). Escrever os números em ordem CRESCENTE.
- *Exercício 010*: Sistema de ordenação de valores. Ler 5 valores (considere que não serão informados valores iguais). Escrever os números em ordem DECRESCENTE.
- *Exercício 011*: Ler x números, onde x é definido pelo usuário (o usuário que decide quando acaba). Escreva o resultado da subtração entre as somas dos números pares e ímpares. Ex: soma dos pares - soma dos ímpares.
- *Exercício 012*: Ler 3 valores e não aceitar valores menores que 1. Caso o usuário digite valor menor que 1, repetir até obter todos os números. Escreva o resultado da soma dos números.

Lógica de Programação

Lógia de Programação - Resumo

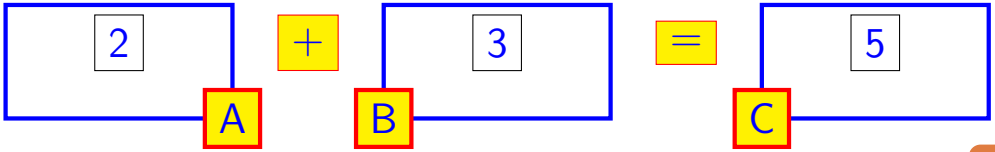
- **Lógica de Programação**:
  - Técnica de encadear pensamentos para atingir determinado objetivo.
  - Necessária para desenvolver programas e sistemas, pois permite definir a sequência lógica para a solução de um problema.
- **Sequência Lógica**:
  - Passos executados até se atingir o objetivo ou solução de um problema.
  - Podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Lógia de Programação - Instrução

- Cada um dos passos, cada uma das ações a tomar (obedecendo a sequência lógica) para ir resolvendo o problema, ou para ir executando a tarefa.
- Em computação, é a informação que indica a um computador uma operação a executar. Ex: somar, subtrair.
- Uma só instrução não resolve problemas reais.
- Executar um conjunto de instruções.
- Executar em uma sequência lógica.

Lógia de Programação - Algoritmo

- Sequência finita de passos que levam à execução de uma tarefa.
- Claro e preciso. Ex: somar dois números:
  - Escrever primeiro número no retângulo A
  - Escrever segundo número no retângulo B
  - Somar os números do retângulo A com o número do retângulo B e escrever o resultado no retângulo C.



Lógia de Programação - Programa

- Algoritmo escrito em uma linguagem de computador (linguagem de programação).
- Interpretado e executado por um computador.
- Interpretação rigorosa, exata pelo computador -> escrita do algoritmo na linguagem de programação tem que seguir regras mais rigorosas.

Lógia de Programação - Plano

Fases para desenvolver o algoritmo:

- Determinar o problema, definí-lo (entender) bem.
- Dividir a solução nas três fases.

Exemplo:

- Problema: calcular a média de quatro números.
- Dados de entrada: os números N1, N2, N3 e N4.
- Processamento: somar os quatro números e dividir a soma por 4.
- Dado de saída: a média final

Vamos praticar

Individualmente cada um escreve o seu algoritmo.

Lógia de Programação - Algoritmo

- Início
- Ler o primeiro número
- Ler o segundo número
- ler o terceiro número
- Ler o quarto número
- Somar todos os números
- Dividir a soma por 4
- Mostrar o resultado da divisão
- Fim

Lógia de Programação - Python

Escreva utilizando Python.

Exercício

Escreva o algoritmo e programa.

- *Exercício 013*: Leia três números inteiros e calcule a soma. Considerar que a condição, se a soma for maior que 10, escreva "tem erro", do contrário escreva o valor resultante da soma.

O que é uma função?

- O conceito de função é um dos mais importantes na matemática.
- Em computação, uma função é uma sequência de instruções que computa um ou mais resultados que chamamos de parâmetros.
- Em aula utilizamos algumas funções já prontas do Python como o print() e input().

O que é uma função?

O Python permite definirmos funções. A sintaxe é muito parecida com a da matemática. Para definirmos uma função no Python utilizamos o comando def:

```
def somar(primeiroNumero, segundoNumero):
    return primeiroNumero + segundoNumero

print('total da soma: '.format(somar(11,20)))
```

Exemplo de função

```python
def somar(primeiro_numero: int, segundo_numero: int) -> int:
  return primeiro_numero + segundo_numero

# testes da classe
#if __name__ == "__main__":
# print('total da soma: '.format(somar(11,20)))
```

Código 1: Exemplo de Código para criação de função em python

```python
from codigo_001_somar import somar
import pytest

@pytest.mark.parametrize(('primeiro_numero', 'segundo_numero', 'resultado_esperado'), [(1, 1, 2), (11, 1, 12)])
def teste_somar_com_sucesso(primeiro_numero, segundo_numero, resultado_esperado):
  assert somar(primeiro_numero, segundo_numero) == resultado_esperado

@pytest.mark.parametrize(('primeiro_numero', 'segundo_numero', 'resultado_esperado'), [(1, 1, 3), (11, 1, 11)])
def teste_soma_com_erro(primeiro_numero, segundo_numero, resultado_esperado):
  assert somar(primeiro_numero, segundo_numero) != resultado_esperado
```

Código 2: Exemplo de Código para criação de dois casos de testes para a função somar

Pytest

```
pip install pytest

pip install pytest-cov

python -m pytest –cov

python -m pytest –cov -q test_filename.py

coverage html && open htmlcov/index.html
```

*obs: são dois traços antes da palavra cov.
*obs: o nome do arquivo que contém os testes deve iniciar com prefíxo test_*, em que o asterisco deve ser substituído pelo "nome do código" que será testado. Exemplo: arquivo test_codigo_001_somar.py utilizado para testar o arquivo codigo_001_somar.py

https://docs.pytest.org/en/7.1.x/contents.html

Pytest

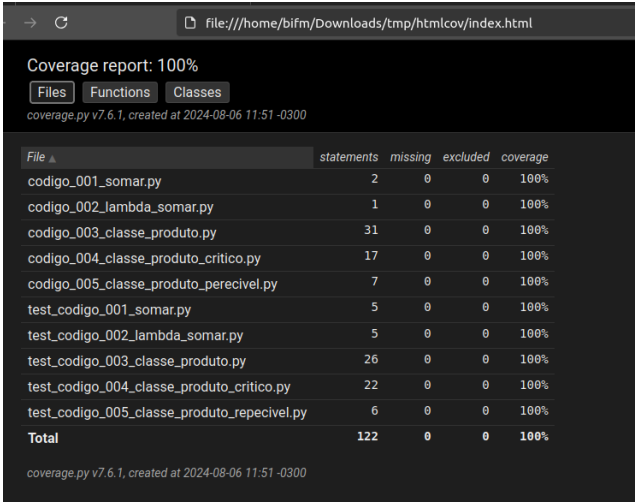
```
$ python3 -m pytest --cov
============================================================================================= test session starts =====
============================================================================================
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0
rootdir: /library/personal/repository/latex/softex-2024-material-backend-python/outros/codigos/python
plugins: metadata-3.1.1, cov-5.0.0
collected 2 items


test_codigo_001_somar.py ..                                                                [100%]


---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                       Stmts   Miss  Cover
----------------------------------------------
codigo_001_somar.py            2      0   100%
test_codigo_001_somar.py       5      0   100%
----------------------------------------------
TOTAL                          7      0   100%


============================================================================================= 2 passed in 0.02s ======
============================================================================================
```

Pytest


file:///home/bifm/Downloads/tmp/htmlcov/index.html
Coverage report: 100%
Files
Functions
Classes
coverage.py v7.6.1, created at 2024-08-06 11:51 -0300
File ▲ statements missing excluded coverage
codigo_001_somar.py 2 0 0 100%
codigo_002_lambda_somar.py 1 0 0 100%
codigo_003_classe_produto.py 31 0 0 100%
codigo_004_classe_produto_critico.py 17 0 0 100%
codigo_005_classe_produto_perecivel.py 7 0 0 100%
test_codigo_001_somar.py 5 0 0 100%
test_codigo_002_lambda_somar.py 5 0 0 100%
test_codigo_003_classe_produto.py 26 0 0 100%
test_codigo_004_classe_produto_critico.py 22 0 0 100%
test_codigo_005_classe_produto_repecivel.py 6 0 0 100%
Total 122 0 0 100%
coverage.py v7.6.1, created at 2024-08-06 11:51 -0300

Exercício

Escreva o programa.

- *Exercício 025*: Leia um número inteiro. Escreva o número lido.

Exercício

Escreva o programa.

- *Exercício 026*: Leia três números inteiro e guarde em uma lista. Escreva os números da lista.

Exemplos com função input()

```python
def obtem_numero() -> int:
    return (int(input("Digite um numero inteiro: ")))

def obter_tres_numeros_simples() -> list:
    return [obtem_numero(), obtem_numero(), obtem_numero()]

def obter_trinta_numeros() -> list:
    resultado: list = []
    while len(resultado) < 30:
        resultado.append(obtem_numero())
    return resultado

def obter_n_numeros(quantidade: int = 90) -> list:
    resultado: list = []
    while len(resultado) < quantidade:
        resultado.append(obtem_numero())
    return resultado
```

Código 3: Código para criação de função com input()

Pytest

```python
from codigo_008_input import obtem_numero
from codigo_008_input import obter_tres_numeros_simples
from codigo_008_input import obter_trinta_numeros
from codigo_008_input import obter_n_numeros
import pytest
from unittest.mock import patch

@pytest.mark.parametrize("valor_esperado", [(10), (9)])
def teste_obter_numero_com_sucesso(valor_esperado):
  with patch('builtins.input', return_value=valor_esperado):
    resultado = obtem_numero()
    assert resultado == valor_esperado

@pytest.mark.parametrize("valor_esperado", [([10,10,10]), ([9,9,9])])
def teste_obter_tres_numeros_simples_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado):
    resultado = obter_tres_numeros_simples()
    assert resultado == valor_esperado

@pytest.mark.parametrize("valor_esperado", [([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]),
      (list(range(0, 30)))])
def teste_obter_trinta_numeros_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado):
    resultado = obter_trinta_numeros()
    assert resultado == valor_esperado

@pytest.mark.parametrize("valor_esperado", [(list(range(0, 90)))])
def teste_obter_n_numeros_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado):
    resultado = obter_n_numeros(len(valor_esperado))
    assert resultado == valor_esperado
```

Código 4: Teste para funções criadas com input()

Pytest

```
python$ python3 -m pytest --cov -q test_codigo_008_input.py
.......                                                                  [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                        Stmts   Miss  Cover
-----------------------------------------------
codigo_008_input.py            14      0   100%
test_codigo_008_input.py       26      0   100%
-----------------------------------------------
TOTAL                          40      0   100%

7 passed in 0.05s
```

Exemplos com função print()

```python
from codigo_008_input import obtem_numero, obter_tres_numeros_simples, obter_trinta_numeros, obter_n_numeros

def print_obtem_numero():
    print(obtem_numero())

def print_obter_tres_numeros_simples():
    print(obter_tres_numeros_simples())

def print_obter_trinta_numeros():
    print(obter_trinta_numeros())

def print_obter_n_numeros(n: int = 90):
    print(obter_n_numeros(n))
```

Código 5: Código para criação de função com print()

Pytest

```python
from codigo_009_print import print_obtem_numero, print_obter_tres_numeros_simples, print_obter_trinta_numeros,
        print_obter_n_numeros
import pytest
from unittest.mock import patch

@pytest.mark.parametrize("valor_esperado", [(10), (9)])
def teste_print_obtem_numero_com_sucesso(valor_esperado):
  with patch('builtins.input', return_value=valor_esperado), patch('builtins.print') as mock_print:
    print_obtem_numero()
    mock_print.assert_called_once_with(valor_esperado)
    assert mock_print.call_count == 1

@pytest.mark.parametrize("valor_esperado", [([10,10,10]), ([9,9,9])])
def teste_print_obtem_tres_numeros_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
    print_obter_tres_numeros_simples()
    mock_print.assert_called_once_with(valor_esperado)
    assert mock_print.call_count == 1

@pytest.mark.parametrize("valor_esperado", [([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]),
      (list(range(0, 30)))])
def teste_print_obtem_trinta_numeros_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
    print_obter_trinta_numeros()
    mock_print.assert_called_once_with(valor_esperado)
    assert mock_print.call_count == 1

@pytest.mark.parametrize("valor_esperado", [(list(range(0, 90)))])
def teste_print_obtem_n_numeros_com_sucesso(valor_esperado):
  with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
    print_obter_n_numeros(len(valor_esperado))
    mock_print.assert_called_once_with(valor_esperado)
    assert mock_print.call_count == 1
```

Código 6: Teste para funções criadas com print()

Pytest

```
python$ python3 -m pytest --cov -q test_codigo_009_print.py
.......

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                       Stmts   Miss  Cover
--------------------------------------------
codigo_008_input.py           14      0   100%
codigo_009_print.py            9      0   100%
test_codigo_009_print.py      27      0   100%
--------------------------------------------
TOTAL                         50      0   100%

7 passed in 0.06s
```

Exercícios

- *Exercício 015*: Leia o número de maçãs compradas, calcule e escreva o custo total da compra. Considere que as maçãs custam R$ 1,50 cada se forem compradas menos de uma dúzia, e R$ 1,00 se forem compradas pelo menos 12.
- *Exercício 016*: A jornada de trabalho semanal de um funcionário é de 40 horas. O funcionário que trabalhar mais de 40 horas receberá hora extra, cujo cálculo é o valor da hora regular com um acréscimo de 50%. Leia o número de horas trabalhadas em um mês, o salário por hora e escreva o salário total do funcionário, que deverá ser acrescido das horas extras, caso tenham sido trabalhadas (considere que o mês possua 4 semanas exatas).

Exemplo de função anônima (lambda)

Funções lambda são uma ferramenta poderosa e versátil na programação Python, que permite criar funções anônimas de forma simples e rápida.

Uma função lambda é criada usando a palavra-chave lambda, seguida de um ou mais argumentos, e uma expressão.

Argumentos são os dados de entrada que esta função irá receber expressão é o código que será executado quando a função lambda for chamada.

Sua sintaxe básica é a seguinte:

```
lambda argumentos: expressão
```

Exemplos de funções anônimas (lambda)

```python
somar = lambda primeiro_numero, segundo_numero: primeiro_numero + segundo_numero

# print('total da soma: '.format(soma(11,20)))

# ou

# print((lambda primeiro_numero, segundo_numero: primeiro_numero + segundo_numero)(11, 20))
```

Código 7: Exemplo de Código para criação de função anônima em python

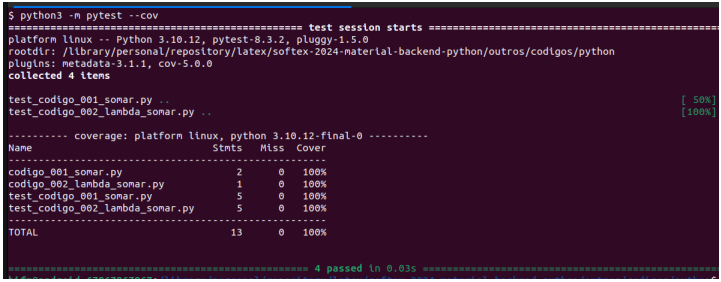
Pytest

```python
from codigo_002_lambda_somar import somar

def teste_somar_com_sucesso():
  assert somar(1,2) == 3

def teste_soma_com_erro():
  assert somar(1,2) != 2
```

Código 8: Exemplo de Código para testar função anônima em python



Exercícios

- *Exercício 017*: Ler 4 números inteiros que correspondem ao número da conta do cliente, saldo, débito ou crédito. Calcular e escrever o saldo atual (saldo atual = saldo - débito + crédito). Também testar se saldo atual for maior ou igual a zero, escrever a mensagem 'Saldo Positivo', senão escrever a mensagem 'Saldo Negativo'.

Padrões de Desenvolvimento de Software

Padrões de Desenvolvimento de Software

...São soluções típicas para problemas comuns em projeto de software.

- Reusabilidade de Software.
- Conceitos e uso de SOLID.
- Conceitos básicos de Padrões de Projeto.
- Padrões de Criação.
- Padrões Estruturais.
- Padrões Comportamentais.
- Padrões Arquiteturais(MVC).

Padrões de Desenvolvimento de Software

Reúso de software é o processo de incorporar produtos existentes em um novo produto.

- Código.
- specificações de Requisitos e Projeto.
- Planos de Teste.
- Conhecimento.

Padrões de Desenvolvimento de Software

Benefícios

- Aumento da Produtividade.
- Diminuição do tempo de desenvolvimento e validação -> Redução de custo.
- Qualidade dos Produtos.
- Flexibilidade na estrutura do software.
- Manutenibilidade.
- Familiaridade com o uso de padrões -> leva a menos erros.

Padrões de Desenvolvimento de Software

Dificuldades

- Identificação e compreensão dos artefatos.
- Qualidade dos artefatos.
- Modificação dos artefatos.
- Falta de confiança nos “artefatos dos outros”. Mito: “não inventado aqui.”.
- Ferramentas de apoio.
- Aspectos legais e econômicos.
- Falta de incentivo.

Padrões de Desenvolvimento de Software

Requisitos

- Catalogação, documentação e certificação completa do artefato a ser reutilizado, de modo a ser possível:
  - Encontrar o artefato a ser reutilizado.
  - Compreender o artefato para adaptá-lo ao novo contexto.
  - Garantir que o artefato se comportará conforme especificado.

Padrões de Desenvolvimento de Software

Uma boa técnica de reúso deve garantir adaptação e adequação a um novo contexto:

- Abstração.
- Seleção.
- Especialização.
- Integração.

Padrões de Desenvolvimento de Software

Técnicas para Reúso.

- Bibliotecas.
- Frameworks.
- Componentes.
- Padrões de Software.
- Linhas de Produto de Software.

Princípios de projetos S.O.L.I.D.

Padrões de Desenvolvimento de Software

SOLID - são cinco princípios de design de **código orientado a objeto** que basicamente tem os seguintes objetivos.

- Tornar o código mais entendível, claro e conciso.
- Tornar o código mais flexível e tolerante a mudanças.
- Aumentar a adesão do código aos princípios da orientação a objetos.

Padrões de Desenvolvimento de Software

SOLID é um acrônimo para cada um dos cinco princípios que fazem parte desse grupo:

- *Single Responsability Principle* (Princípio da Responsabilidade Única).
- *Open/Closed Principle* (Princípio do "Aberto para Extensão/Fechado para Implementação).
- *Liskov Substitution Principle* (Princípio da Substituição de Liskov).
- *Interface Segregation Principle* (Princípio da Segregação de Interfaces).
- *Dependency Inversion Principle* (Princípio da Inversão de Dependências).

Padrões de Desenvolvimento de Software

Single Responsability Principle (Princípio da Responsabilidade Única).

- Uma classe deve ter UM, e somente um, MOTIVO para mudar.
- Alterações causam "incerteza"
  - Cada linha modificada pode introduzir BUG novo.
  - Diminui a coesão e aumenta acoplamento.
- É o padrão "base".
- Anti-padrões: Classe Deus/Grande bola de lama/etc.

Padrões de Desenvolvimento de Software

Open/Closed Principle (Princípio do "Aberto para Extensão/Fechado para Implementação).

- Objetos e/ou classes devem estar abertos para extensão, mas fechados para modificação.
  - É possível incluir novas funcionalidades.
- Alterações causam "incerteza" (de novo).
- Abstração é a chave
- Utilização de bom encapsulamento.
  - Atributos sempre privados.
  - Uso de polimorfismo: criação de interfaces/classes abstratas.
  - Jamais usar variáveis globais ou "similares".
- Anti-padrões: código espaguete.

Padrões de Desenvolvimento de Software

Liskov Substitution Principle (Princípio da Substituição de Liskov).

- Uma classe derivada deve ser substituível pela sua classe base.
  - "Se para cada objeto o1 do tipo S há um objeto o2 do tipo T de forma que, para todos os programas P definidos em termos de T, o comportamento de P é inalterado quando o1 é substituído por o2 então S é um subtipo de T.
- Utilização de poliformismo adequado.
  - A validade do modelo depende de seus filhos.
  - Relacionamento IS-A ligado ao comportamento.
    - Problemas em CASTs.
- Pode ser relacionado com "Design por contrato".
  - Pré condições e pós condições na execução.

Padrões de Desenvolvimento de Software

Interface Segregation Principle (Princípio da Segregação de Interfaces).

- Classe não implementa interface com métodos que não vai usar.
- Evitar poluição da interface -> baixo acoplamento.
- Anti-padrão: interface "Deus"/martelo de ouro.

Padrões de Desenvolvimento de Software

Dependency Inversion Principle (Princípio da Inversão de Dependências).

- Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.
- Abstrações não devem depender de detalhes. Detalhes (implementações) devem depender de abstrações.
- Utilização constante de polimorfismo.
  - Facilita a reutilização.
- Anti-padrão: complexidade "acidental"/gambiarra/copiar-colar.

POO

Programação Orientada a Objetos (POO)

Tradicionalmente a programação de sistemas considera os dados separados das funções.

- Os dados são estruturados de modo a facilitar a sua manipulação pelas funções, mas as funções estão livres para usar os dados como quiserem.
- Os aspectos segurança e integridade dos dados ficam de certa forma vulneráveis.
- A programação tradicional de sistemas tem o seu mais forte e bem-sucedido modelo na **programação estruturada**.

Programação Orientada a Objetos (POO)

Por outro lado, na programação orientada a objetos (POO), os dados específicos do objeto são estruturados junto com as funções que operam sobre esses dados.

- Linguagem como Python adota esse paradigma, onde os objetos são instâncias de classes, constituídas por variáveis (atributos) e métodos (funções) que operam sobre esses dados.
- A POO busca robustez, adaptabilidade e reusabilidade, e seus princípios incluem modularidade, abstração e encapsulamento.
- No desenvolvimento de software, o projeto, a implementação e os testes são etapas essenciais, e a definição clara das classes e suas responsabilidades é fundamental para o sucesso do sistema.

Programação Orientada a Objetos (POO)

Na Programação Orientada a Objetos (POO), os dados específicos do objeto são estruturados juntamente com as funções que são permitidas sobre esses dados. Essa forma de programar é vista na linguagem Java.

Programação Orientada a Objetos (POO)

Os principais elementos da POO são os objetos. Dizemos que um objeto é uma instância de uma classe. Uma **Classe** é constituída por variáveis (membros de dados) e métodos ou funções (membros da função).

- A Classe é o modelo. Um objeto é um elemento deste modelo.
- As variáveis de uma Classe são também chamadas de **Atributos**.
- As funções de uma Classe são também chamadas de **Métodos**.

Objetivos da POO

- Robustez – o programa não pode cair frente a dados inesperados.
- Adaptabilidade – rodar facilmente em diferentes ambientes.
- Reusabilidade – usar os elementos já construídos em outros sistemas.

Princípios da POO

- Modularidade – dividir o sistema em pequenas partes bem definidas.
- Abstração – identificar as partes fundamentais (tipos de dados e operações), definindo o que cada operação faz e não necessariamente como é feito.
- Encapsulamento – a interface para uso de cada componente deve estar bastante clara para todos que usam esse componente. Detalhes internos de implementação não interessam.

O desenvolvimento de software

- Há apenas diretrizes gerais que quando usadas levam em geral a um bom resultado.
- O desenvolvimento certamente é influenciado pelo ambiente computacional no qual será desenvolvido.
- A linguagem de programação e o ambiente ou plataforma na qual será desenvolvido o sistema podem decidir o modo, a estratégia e os passos que serão usados.

O desenvolvimento de software

Podemos dividir o desenvolvimento de software em 3 etapas, independente da linguagem, sistema operacional ou plataforma de desenvolvimento que será usada.

- O projeto.
- A implementação.
- Os testes e depuração.

O desenvolvimento de software

O projeto é a parte mais importante. Nele são definidas as classes e a relação entre elas. Os princípios que devem orientar a definição das classes são:

- **Responsabilidade de cada uma delas** – quais problemas elas resolve.
- **Independência entre elas** – se uma precisa da outra e vice-versa.
- **Comportamento** – quais as entradas (parâmetros) e saídas (resultados) das classes.

https://docs.python.org/pt-br/3/tutorial/classes.html

Resumo sobre Programação Orientada a Objetos

Os princípios básicos da POO são descritos a seguir.

- **Classe**: Representação de um conjunto de objetos com características afins. Definição do comportamento dos objetos (métodos) e seus atributos.
- **Objeto**: Uma instância de uma classe. Armazena estados por meio de atributos e reação a mensagens enviadas por outros objetos.
- **Abstração**: Oculta detalhes que não são necessários no contexto.
- **Herança**: Mecanismo pela qual uma classe (sub-classe) pode estender outra classe (super-classe), estendendo seus comportamentos e atributos.
- **Polimorfismo**: Princípio pelo qual as instâncias de duas classes ou mais classes derivadas de uma mesma super-classe podem invocar métodos com a mesma assinatura, mas com comportamentos distintos.
- **Encapsulamento**: Proibição do acesso direto ao estado de um objeto, disponibilizando apenas métodos que alterem esses estados na interface pública.

Um exemplo de programação orientada a objeto com Python

Vamos desenvolver um sistema que possui produtos. O objeto em questão é o produto. Assim, vamos criar a classe Produto, que tornará possível construir ou criar elementos ou objetos desta classe.

A sintaxe básica para criação de uma classe é´:

```
class NomeDaClasse:
    Instruções
```

- **classe** - Produto.
- **atributos** - nome, codigo, preco, quantidade.
- **métodos** - obtem_nome, obtem_codigo, obtem_preco, altera_preco, altera_quantidade.

Requisitos da classe Produto

Vamos listar abaixo:

- Deve ser possível recuperar o nome, código e preço do produto.
- Devolve True se novo preço for maior que o atual preço.
- Devolve False se a quantidade de produtos requerida não está disponível.

Atenção

Exemplos de criação de classe

```python
class Produto:
    serie: int # público
    __lote: int # privado
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
        self.nome = nome
        self.codigo = codigo
        self.preco = preco
        self.quantidade = quantidade
        self.serie = round(codigo * 1000)
        self.__lote = round(1000)
    def obtem_nome(self)-> str:
        return self.nome
    def obtem_codigo(self)-> int:
        return self.codigo
    def obtem_preco(self)-> int:
        return self.preco
    def obtem_serie(self)-> int:
        return self.serie
    def obtem_lote(self)-> int:
        return self.__lote
    def altera_preco(self, novo_preco) -> bool:
        preco_atual = self.preco
        self.preco = novo_preco
        if novo_preco > preco_atual: return True
        return False
    def altera_quantidade(self, novo_pedido: int) -> bool:
        if novo_pedido > self.quantidade: return False
        self.quantidade -= novo_pedido
        return True
    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

Código 9: Código da classe Produto

Exemplos de criação de classe

Se o código abaixo for colocado junto com o arquivo da classe Produto, é posssível "testar" o código no mesmo arquivo. É desaconselhável fazer.

```
if __name__ == "__main__":
    p1 = Produto("Laranja", 1, 1.56, 10)
    print("Oferta do dia:", p1.obtem_nome())
    if p1.altera_preco(40.00): print("Preco alterado hoje")
    else: print("Atencao - baixou o preco")
```

```python
from codigo_003_classe_produto import Produto
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def testa_criacao_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
      produto.obtem_lote() != 0 and produto.obtem_serie() != 0

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def teste_alteracao_preco_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.altera_preco(2) == True

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def teste_alteracao_preco_de_produtos_com_falha(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.altera_preco(1) == False

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def teste_alteracao_quantidade_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.altera_quantidade(2) == False

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def teste_alteracao_quantidade_de_produtos_com_falha(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.altera_quantidade(1) == True

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade):
  produto = Produto(nome, codigo, preco, quantidade)
  assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo,
      preco, quantidade)
```

Código 10: Cobertura de testes da classe Produto

Pytest

```
$ python3 -m pytest --cov -q test_codigo_003_classe_produto.py
..........                                                               [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                    Stmts   Miss  Cover
-----------------------------------------------------------
codigo_003_classe_produto.py               21      0   100%
test_codigo_003_classe_produto.py          22      0   100%
-----------------------------------------------------------
TOTAL                                      43      0   100%

10 passed in 0.03s
```

classe Produto

O nome **self** refere-se ao particular objeto sendo criado. Note que o primeiro parâmetro é sempre self na definição. No uso ou na chamada do método esse primeiro parâmetro não existe.

- No exemplo anterior incluímos além da definição da classe, alguns comandos de teste dos métodos da mesma. Assim o módulo (arquivo onde está armazenada a classe) poderia chamar-se Produto.py.
- O comando if **__name__** == "**__main__**" usado antes dos testes anteriormente, determina que os comandos abaixo somente serão executados quando a classe estiver sendo testada, isto é, quando for solicitada a execução do módulo Produto.py

Uso e declaração dos métodos

Quando o método é declarado sempre o primeiro parâmetro é self, quando é necessário acesso ao objeto. Entretanto nas chamadas omite-se esse parâmetro. O correspondente ao self torna-se o prefixo da chamada.

Declaração:

```
def altera_preco(self, novo_preco):
    Instruções
```

Uso:

```
p1 = Produto("Arroz", 1, 2, 5)
if p1.altera_preco(2): print('False')
```

O método construtor

__init__ é um método especial dentro da classe. É construtor da classe, onde os atributos do objeto recebem seus valores iniciais.

- No caso da classe acima, os 4 atributos (variáveis) que caracterizam um produto, recebem neste método seus valores iniciais, que podem ser modificados por operações futuras deste mesmo objeto.
- A criação de um novo produto, ou seja, a criação de uma instância do objeto produto causa a execução do método **__init__**. Assim, o comando abaixo, causa a execução do método **__init__**:

```
p1 = Produto("Arroz", 1, 2, 5)
```

Explorar a sintaxe das classes – o parâmetro self

O parâmetro self é importante no método construtor da classe **__init__**, para referenciar o objeto que está sendo criado. Nos demais métodos, se usado, é simplesmente um parâmetro. Nem precisa ser o primeiro. Mesmo no **__init__**, o primeiro parâmetro pode ter qualquer nome.

```
class Produto:
    def __init__(outro, nome):
        outro.nome = nome
    def altera_nome(nome, self):
        self.nome = nome
```

Explorar a sintaxe das classes – o parâmetro self

- No caso da classe acima, os 4 atributos (variáveis) que caracterizam um produto, recebem neste método seus valores iniciais, que podem ser modificados por operações futuras deste mesmo objeto.
- A criação de um novo produto, ou seja, a criação de uma instância do objeto produto causa a execução do método **__init__**. Assim, o comando abaixo, causa a execução do método **__init__**:

```
p1 = Produto("Arroz", 1, 2, 5)
```

Explorar a sintaxe das classes

Uma classe não precisa necessariamente ter um método construtor. Podemos ter uma classe apenas com métodos, sem atributos. Nesse caso, o nome da classe é usado sem parâmetros para a chamada das funções.

```
class Produto:

    def imprime_nome(nome):

        print(nome)


Produto.imprime_nome('Arroz')
```

Herança em classes

Permite que uma classe seja definida com base em classe já existente. Dizemos que essa nova classe herda características da classe original.

- Na terminologia Python a classe original é chamada de Classe Base, Classe Mãe ou Superclasse (Base, Parent ou Super Class) enquanto que a nova é chamada de Sub Classe ou Classe Filha (Sub ou Child Class).
- A subclasse pode especializar a classe principal ou mesmo estendê-la com novos métodos e atributos.

Herança em classes

Uma classe não precisa necessariamente ter um método construtor. Podemos ter uma classe apenas com métodos, sem atributos. Nesse caso, o nome da classe é usado sem parâmetros para a chamada das funções.

```
class Produto:

    def imprime_nome(nome):

        print(nome)


Produto.imprime_nome('Arroz')
```

Herança em classe

```python
from codigo_003_classe_produto import Produto

class ProdutoCritico(Produto):
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, estoque_min: int):
        super().__init__(nome, codigo, preco, quantidade)
        self.estoque_min = estoque_min

    def obtem_serie(self)-> int:
        return self.serie

    def altera_quantidade(self, novo_pedido: int) -> bool:
        if novo_pedido + self.estoque_min > self.quantidade:
            return False
        return super().altera_quantidade(novo_pedido)

    def altera_preco(self, novo_preco: int) -> bool:
        preco_atual = self.preco
        if novo_preco > 1.1 * preco_atual or novo_preco < 0.9 * preco_atual:
            return False
        super().altera_preco(novo_preco)
        return True

    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}'.format(self.nome, self.codigo, self.preco, self.quantidade, self.estoque_min)
```

Código 11: Herança da classe Produto

*obs: **class ProdutoCritico(Produto)** é uma classe derivada de Produto: indica que **ProdutoCritico** é uma subclasse da classe **Produto**.

Pytest

```python
from codigo_004_classe_produto_critico import ProdutoCritico
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
def teste_alteracao_preco_de_produtos_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert produto_critico.altera_preco(11) == True

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
def teste_alteracao_preco_de_produtos_com_falha(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert produto_critico.altera_preco(12) == False

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
def teste_alteracao_quantidade_de_produtos_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert produto_critico.altera_quantidade(2) == False

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
def teste_alteracao_quantidade_de_produtos_com_falha(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert produto_critico.altera_quantidade(1) == True

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert  produto_critico.obtem_produto() == '{}-{}-{}-{}-{}'.format(nome, codigo, preco, quantidade, estoque_min)

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
def teste_obtem_serie_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
  produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
  assert  produto_critico.obtem_serie() == produto_critico.serie
```

Código 12: Cobertura de testes da classe Produto Crítico

Pytest

```
$ python3 -m pytest --cov
============================================================================
============================================================
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0
rootdir: /library/personal/repository/latex/softex-2024-material-backe
plugins: metadata-3.1.1, cov-5.0.0
collected 23 items


test_codigo_001_somar.py ..
                                                          [  8%]
test_codigo_002_lambda_somar.py ..
                                                          [ 17%]
test_codigo_003_classe_produto.py ...........
                                                          [ 69%]
test_codigo_004_classe_produto_critico.py ......
                                                          [ 95%]
test_codigo_005_classe_produto_perecivel.py .
                                                          [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                     Stmts   Miss  Cover
----------------------------------------------------------------------------
codigo_001_somar.py                                          2      0   100%
codigo_002_lambda_somar.py                                   1      0   100%
codigo_003_classe_produto.py                                31      0   100%
codigo_004_classe_produto_critico.py                        19      0   100%
codigo_005_classe_produto_perecivel.py                       7      0   100%
test_codigo_001_somar.py                                     5      0   100%
test_codigo_002_lambda_somar.py                              5      0   100%
test_codigo_003_classe_produto.py                           26      0   100%
test_codigo_004_classe_produto_critico.py                   26      0   100%
test_codigo_005_classe_produto_perecivel.py                  6      0   100%
----------------------------------------------------------------------------
TOTAL                                                      128      0   100%
```

Polimorfismo em classe

Polimorfismo permite que objetos de diferentes classes sejam tratados como objetos de uma classe comum.

- Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação, assinatura, mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.

Polimorfismo em classe

```python
from codigo_003_classe_produto import Produto

class ProdutoPerecivel(Produto):
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, validade: int):
        super().__init__(nome, codigo, preco, quantidade)
        self.validade = validade

    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}'.format(self.nome, self.codigo, self.preco, self.quantidade, self.validade)
```

Código 13: Polimorfismo de classe

*obs: **class ProdutoPerecivel(Produto)** é uma classe derivada de Produto: indica que **ProdutoPerecivel** é uma subclasse da classe **Produto**.

Pytest

```python
from codigo_005_classe_produto_perecivel import ProdutoPerecivel
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'validade'), [('Arroz', 1, 10, 100, 20)])
def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, validade):
  produto_perecivel = ProdutoPerecivel(nome, codigo, preco, quantidade, validade)
  assert  produto_perecivel.obtem_produto() == '{}-{}-{}-{}-{}'.format(nome, codigo, preco, quantidade, validade)
```

Código 14: Cobertura de testes da classe Produto Perecivel

Pytest

```
python3 -m pytest --cov -q test_codigo_005_classe_produto_repecivel.py
.

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                             Stmts   Miss  Cover
--------------------------------------------------------------------
codigo_003_classe_produto.py                        23     11    52%
codigo_005_classe_produto_perecivel.py               7      0   100%
test_codigo_005_classe_produto_repecivel.py          6      0   100%
--------------------------------------------------------------------
TOTAL                                               36     11    69%

1 passed in 0.02s
```

Pytest

```
python3 -m pytest --cov
============================================================================= test session starts
=====================================================
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0
rootdir: /library/personal/repository/latex/softex-2024-material-backend-python/outros/codigos/py
plugins: metadata-3.1.1, cov-5.0.0
collected 22 items


test_codigo_001_somar.py ..
                                                                        [  9%]
test_codigo_002_lambda_somar.py ..
                                                                        [ 18%]
test_codigo_003_classe_produto.py ...........
                                                                        [ 72%]
test_codigo_004_classe_produto_critico.py .....
                                                                        [ 95%]
test_codigo_005_classe_produto_repecivel.py .
                                                                        [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                           Stmts   Miss  Cover
----------------------------------------------------------------------------------
codigo_001_somar.py                                                2      0   100%
codigo_002_lambda_somar.py                                         1      0   100%
codigo_003_classe_produto.py                                      23      0   100%
codigo_004_classe_produto_critico.py                              17      0   100%
codigo_005_classe_produto_perecivel.py                             7      0   100%
test_codigo_001_somar.py                                           5      0   100%
test_codigo_002_lambda_somar.py                                    5      0   100%
test_codigo_003_classe_produto.py                                 26      0   100%
test_codigo_004_classe_produto_critico.py                         22      0   100%
test_codigo_005_classe_produto_repecivel.py                        6      0   100%
----------------------------------------------------------------------------------
TOTAL                                                            114      0   100%


============================================================================= 22 passed in 0.10s =
```

Polimorfismo em classe

Sobreposição.

- Sobreposição (Override): é sobrescrever, ou seja, definir um novo comportamento para um método que já existe. Isso acontece quando a classe em questão herda (estende) outra classe e se cria um método com a mesma assinatura da classe "pai" na classe filha.

Encapsulamento em classe

Encapsulamento é a proteção dos atributos ou métodos de uma classe, em Python existem somente o public e o private e eles são definidos no próprio nome do atributo ou método.

```
class Veiculo:
    chassi = 1 # atributo publico
    __motor = 2 # atributo privado a classe Veiculo. O símbolo __* define como privado.

class Carro(Veiculo):
    __placa = 3 # atributo privado a classe Carro

    def __init__(self):
        print(self.chassi)
        print(self.__placa)

veiculo = Veiculo()
print(veiculo.chassi) # imprime 1

carro = Carro() # Erro
# print(carro.__motor) # Erro, pois __motor é privado a classe Veiculo.
# print(carro.__placa) # Erro, __placa é um atributo privado, somente chamado pela classe Carro.
```

Encapsulamento em classe

Exemplo anteior

```
class Veiculo:
    chassi = 1 # atributo publico
    __motor = 2 # atributo privado a classe Veiculo. O símbolo __* define como privado.

class Carro(Veiculo):
    __placa = 3 # atributo privado a classe Carro

    def __init__(self):
        print(self.chassi)
        print(self.__placa)

veiculo = Veiculo()
print(veiculo.chassi) # imprime 1

carro = Carro() # Erro
# print(carro.__motor) # Erro, pois __motor é privado a classe Veiculo.
# print(carro.__placa) # Erro, __placa é um atributo privado, somente chamado pela classe Carro.
```

Tratamento de exceções

O tratamento de exceção, na ciência da computação, é o mecanismo responsável pelo tratamento da ocorrência de condições que alteram o fluxo normal da execução de programas de computadores.
Para condições consideradas parte do fluxo normal de execução, ver os conceitos de sinal e evento.

Saiba mais em https://docs.pytest.org/en/stable/how-to/assert.html

Tratamento de exceções

A sintaxe básica é:

```
try:
   Instruções # o código da funcionalidade.
...
except <ExceptType>:
   Instruções # o código para tratamento da exceção.
...
finally: # Caso o fluxo não seja interrompido, sempre é executado o finally.
   Instruções
```

https://docs.python.org/pt-br/3/tutorial/errors.html

Tratamento de exceções

```python
class Veiculo:
    chassi: int # público
    __motor: int # privado
    def __init__(self, chassi: int, motor: int):
        self.chassi = chassi
        self.__motor = motor
    def obtem_chassi(self)-> int:
        return self.chassi
    def obtem_motor(self)-> int:
        return self.__motor
    def altera_chassi(self, novo_chassi: int) -> bool:
        try:
            if novo_chassi != self.chassi:
                self.chassi = novo_chassi
                return True
            else:
                raise ValueError("Chassi igual ao atual")
        except ValueError as error:
            raise ValueError(error)
```

Código 15: Tratamento de exceções

Pytest

```python
from codigo_006_classe_veiculo_com_exception import Veiculo
import pytest

@pytest.mark.parametrize(('chassi', 'motor'), [('KHG969878976G7DF9G7', 'SGD97S9')])
def teste_obtem_chassi_e_motor_com_sucesso(chassi, motor):
  veiculo = Veiculo(chassi, motor)
  assert  veiculo.obtem_chassi() == chassi and veiculo.obtem_motor() == motor

@pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G7')])
def teste_alterar_chassis_iguais(chassi, motor, novo_chassi):
  veiculo = Veiculo(chassi, motor)
  with pytest.raises(ValueError) as excinfo:
    veiculo.altera_chassi(novo_chassi)
  assert str(excinfo.value) == "Chassi igual ao atual" and excinfo.type is ValueError

@pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G6')])
def teste_alterar_chassis_diferentes(chassi, motor, novo_chassi):
  veiculo = Veiculo(chassi, motor)
  assert veiculo.altera_chassi(novo_chassi) == True
```

Código 16: Cobertura de testes da classe Veiculo

Padrões de Desenvolvimento de Software

Padrões de Desenvolvimento de Software

...São soluções típicas para problemas comuns em projeto de software.

- Reusabilidade de Software.
- Conceitos e uso de SOLID.
- Conceitos básicos de Padrões de Projeto.
- Padrões de Criação.
- Padrões Estruturais.
- Padrões Comportamentais.
- Padrões Arquiteturais(MVC).

Backend
Python com Django

Prof. Bruno Iran Ferreira Maciel

BRUNO

Aug 2024

Exercícios 1

Referências

Referências 1

G. Ghetan. *Aprendendo Padrões de Projeto em Python*. Novatec, 1 edition, 2016.