Backend

Python com Django

Prof. Bruno Iran Ferreira Maciel

Aug 2024

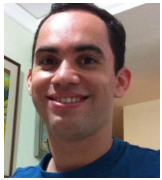
Estrutura de Tópicos das Aulas

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.** Ghetan [2016]

Acrônimo 1

CVCS *Centralized Version Control Systems (em português, Sistema Centralizado de Controle de Versão)* 105
DVCS *Distributed Version Control Systems (em português, Sistema Distribuído de Controle de Versão)* 105
GUI *Graphical User Interface (em português, Interface Gráfica do Utilizador)* . . . 145
LVCS *Local Version Control Systems (em português, Sistema Local de Controle de Versão)* 105
MVC *Model-View-Controller (em português, Modelo-Visão-Controle ou também como Controlador de visualização de modelo)* 139, 143–145
POO *Programação Orientada a Objetos* . . . 81, 82, 86, 98, 154, 164, 178, 180, 186
VCS *Version Control Systems (em português, Sistema de Controle de Versão)* 104, 106, 107

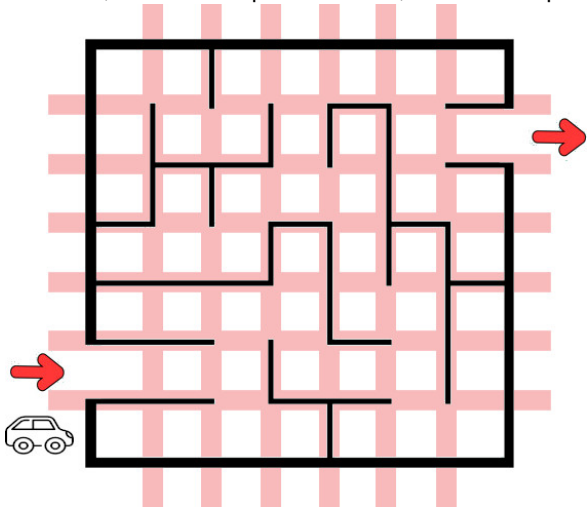
Apresentação e boas-vindas

Apresentação e boas-vindas

Olá?

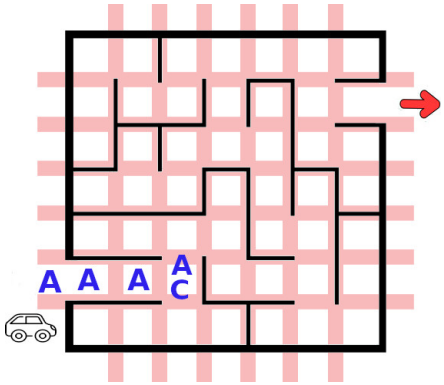
Desafio

* 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

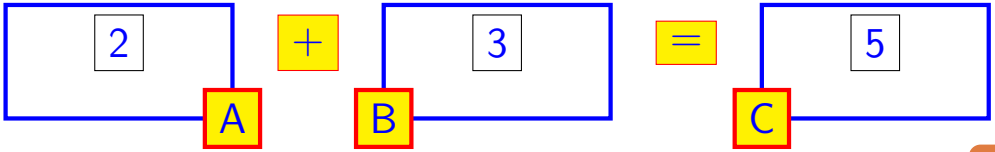
- **Lógica de Programação**:
  - Técnica de encadear pensamentos para atingir determinado objetivo.
  - Necessária para desenvolver programas e sistemas, pois permite definir a sequência lógica para a solução de um problema.
- **Sequência Lógica**:
  - Passos executados até se atingir o objetivo ou solução 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 src 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

Pytest

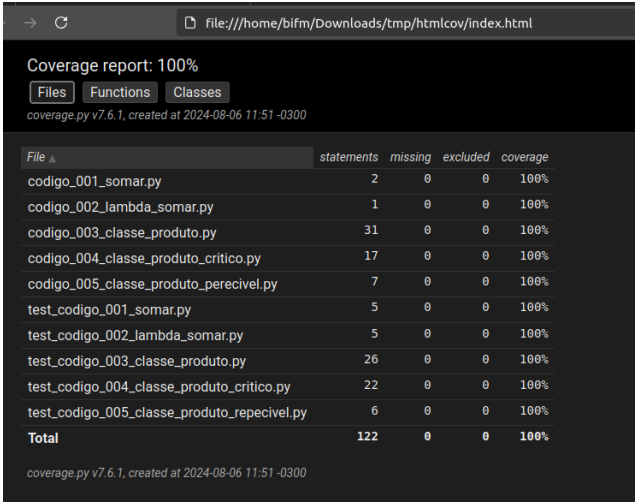
```
$ python3 -m pytest --cov
============================================================================================= test session starts ======
============================================================================================
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0
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 src import obtem_numero, obter_tres_numeros_simples, obter_trinta_numeros, 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 src.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 src 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

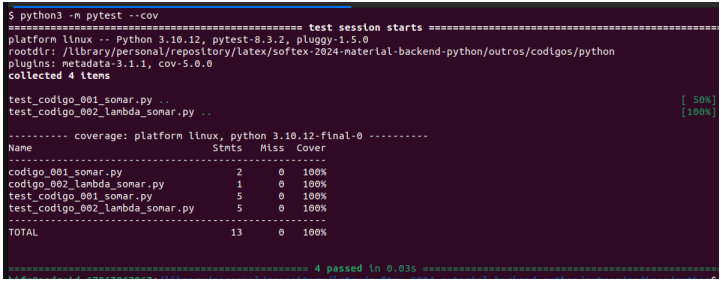
Pytest

```python
from src import somar as somar_lambda

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

def teste_soma_com_erro():
  assert somar_lambda(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. Os número serão passados na inicialização do script. Calcular e escrever o saldo atual (saldo atual = saldo - débito + crédito). Também verificar 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**.

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).

POO

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.

Python Enhancement Proposals (PEP, em português, Propostas de Melhoria do Python) 8 - Guia de Estilo Para Python - pt-br

PEP 8 – Style Guide for Python Code - inglês

*obs: PEP é um conjunto de convenções e práticas para a linguagem de código Python, essas práticas tem uso difundido na comunidade Python e facilitam a compreensão e edição do seu código.

Objetivos da POO

- Eles representam entidades em sua aplicação em desenvolvimento.
- As entidades interagem entre si para resolver problemas do mundo real.
- Por exemplo, Produto é uma entidade, Carro tamém é uma entidade.

Classes

As classes ajudam os desenvolvedores a representar entidades do mundo real:

- As classes definem objetos com atributos e comportamentos. Os atributos são membros de dados e os comportamentos são manifestados pelas funções-membro.
- As classes são constituídas de construtores que proporcionam o estado inicial para esses objetos.
- As classes são como templates, portanto, podem ser facilmente reutilizadas.

*obs: Por exemplo, a classe Produto tem atributos nome e quantidade e uma função-membro que altera a quantidade do produto.

Vamos praticar

Vamos praticar - Algoritmo

- Início
- Abrir um bloco de notas (qualquer recurso para escrever).
- Escolha um objeto (pode ser algo do seu dia a dia).
- Escreva duas das principais características desse objeto.
- Escreva duas das principais ações desse objeto.
- Escreva em um bloco de notas.
- Salve em arquivo que vamos utilizar até o final da aula de hoje.
- Fim

Vamos praticar - Algoritmo

Solução...

- Objeto escolhido: Carro
- Duas características:
  - Cor
  - Modelo
- Duas ações:
  - Acelerar
  - Frear

Vamos praticar - Algoritmo

Solução...

- Objeto escolhido: Produto **(classe)**
- Duas características: **(atributos)**
  - Preço
  - Quantidade
- Duas ações: **(métodos)**
  - Alterar Preço
  - Alterar Quantidade

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.

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.
- **Composição**: É uma maneira de combinar objetos ou classes. Na composição, um objeto é usado para chamar função-membro sem o uso de herança.

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.

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

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

Git

Git

O que é git?

- É um Version Control Systems (em português, Sistema de Controle de Versão) VCS.
- Esse sistema guarda as mudanças feitas em arquivo ou em uma coleção de arquivos no decorrer do tempo...
- ...assim você pode ir para uma versão arbitrária do arquivo...
- ...e caso algo dê errado, você pode voltar e usar uma versão anterior.

Git - Tipos de VCS

- Local Version Control Systems (em português, Sistema Local de Controle de Versão) LVCS
  - Como o próprio nome diz, é local.
  - Não é possível colaborar com outros desenvolvedores.
  - RCS (Revision Control System (em português, Sistema de Controle de Revisão)) foi um dos VCS locais mais populares
- Centralized Version Control Systems (em português, Sistema Centralizado de Controle de Versão) CVCS
  - VCS, Subversion, Perforce.
  - Um servidor mantém todos os arquivos.
  - Ponto único de falha (e se o servidor cair?).
- Distributed Version Control Systems (em português, Sistema Distribuído de Controle de Versão) DVCS
  - Git, Mercurial, Bazaar, Darcs.
  - Cada cliente tem uma cópia de todo repositório.

Por que aprender Git?

- Desenvolvedores profissionais utilizam Git.
- Principal solução para VCS disponível no mercado.
- Boa parte dos empregos na área exigem conhecimento em Git.

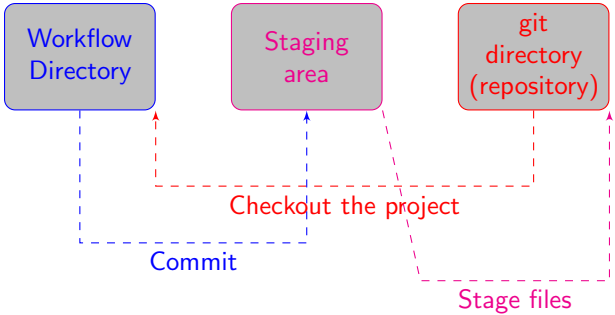
Como o Git funciona?

- Ao invés de armazenar as diferenças (diffs), o Git armazena uma "cópia" dos dados atuais (snapshots).
  - Graças a isso, o Git parece mais um sistema de arquivos que um VCS.
- Quase toda a operação é local.
  - Isso aumenta o desempenho e permite trabalhar offline.
- Checagem de integridade.
  - Praticamente impossível fazer uma alteração sem que o Git fique sabendo.
- Git, geralmente, só adiciona dados.

Os três estados

Um arquivo pode ter três estados.

- unmodified / committed (não modificado / comprometido)
- modified (modificado)
- staged (estágio)



Configurando o Git

1.6 Começando - Configuração Inicial do Git <- clique aqui

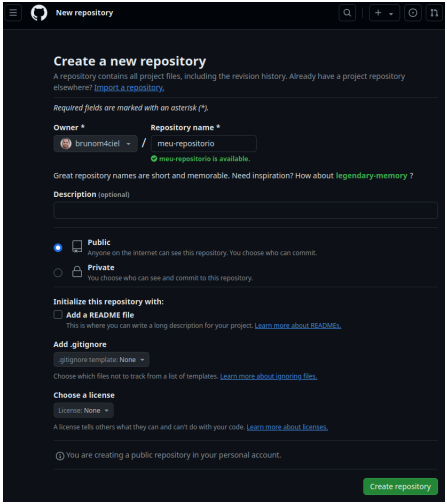
Procure orientação de configuração com base no seu sistema operacional.

Mãos a obra

A solução git que vamos utilizar será o github.

Criar novo repositório <- clique aqui

Criando seu repositório



Clonando um repositório existente

- Início
- Escolha um diretório para guardar o repositório. (procure um local que não tenha outros dados)
- Abra o terminal.
- Vá até o diretório escolhido.
- Execute o comando: $ git clone git@github.com:brunom4ciel/material-python.git
- Executabdo com sucesso deve haver um diretório chamado material-python
- Fim

Fazendo mudanças

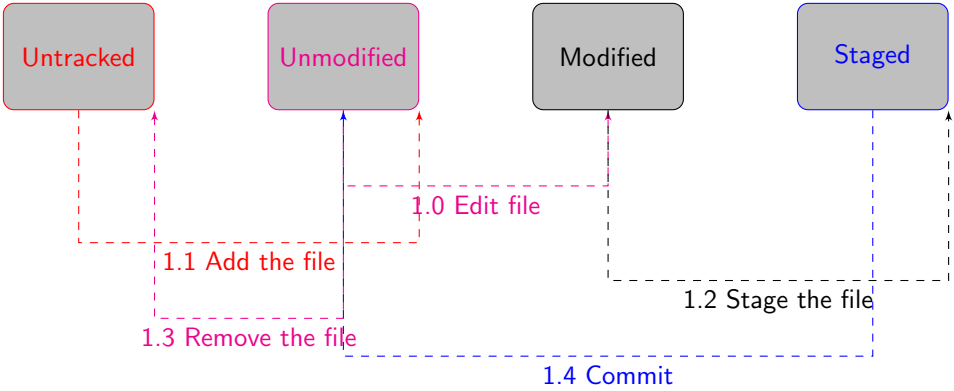
- Cada arquivo pode estar em dois estados: tracked (monitorado) e untracked (não monitorado).
- Arquivos tracked são arquivos que estavam no último snapshot, eles podem estar em qualquer um dos três estados.
- Arquivos untracked são arquivos novos que ainda não estão sendo monitorados.

Adicionando arquivos e o primeiro commit

```
$ git add . -A
$ git commit -m 'versão inicial'
```

- Com isso criamos um repositório local e mesmo sem a ajuda de um servidor externo podemos manter o histórico dos nossos arquivos. Ou seja, é possível usar o Git como LVCS.

Ciclo de vida de um arquivo



Checando o estado dos seus arquivos

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

- se não houver modificações, o repositório está limpo.

Vamos fazer alguns testes

- Adicione um arquivo ao repositório. Digite $ git status e veja o resultado.
- *$ git add . -A* para incluir o arquivo modificado, use *$ git status*. Veja o resultado.
- Modifique o arquivo mais uma vez e *$ git status*. O que aconteceu?

O que aconteceu?

- O Git faz um snapshot do arquivo na hora que você dá um *$ git add*.
- Se você commitar agora, você irá commitar a versão antiga do arquivo, do jeito que ele estava no último *$ git add* feito.
- É necessário outro *$ git add* para que as alterações sejam feitas.

Vendo o histórico de commits

$ git log

commit a7df3cbe37cc9762c9bc4fa16853d459889c8706 (HEAD -> main, origin/main, origin/HEAD)
Author: Avneesh Agarwal <avneeshagarwal0612@gmail.com>
Date: Wed Nov 10 12:05:12 2021 +0530
Delete README.md

commit 10d0879b76f21460a36bf8d246f95f853c08c78a

Author: Your Name <avneeshagarwal0612@gmail.com>
Date: Thu Sep 23 11:59:25 2021 +0530
upgrade packages

Modificando seu último commit

- Alterando só a mensagem do commit:

```
$ git commit –amend
```

- Adicionando um arquivo ao último commit:

```
$ git commit -m 'initial commit'
$ git add README.md
$ git commit –amend
```

Removendo um arquivo staged

- *$ git reset HEAD README.md*

Repositórios remoto

- *$ git remote -v* mostra as URLs do repositório.

Puxando informações do novo repositório

- Um *$ git pull* faz um fetch e depois um merge, incluindo as modificações feitas no repositório pb no seu repositório local.

Enviando suas modificações ao repositório

- *$ git push origin main*

*obs: Você precisa de acesso a escrita para poder enviar informações num repositório remoto.

O que são branchs?

- *$ git branch testing*

*obs: Agora criamos um novo branch, que aponta para a última modificação feita.

Como o Git sabe qual é o branch atual?

- O ponteiro **HEAD** indica qual o branch que está sendo usado.
- Para mudar o HEAD de lugar, usamos: *$ git checkout testing*

Merge

- Vamos fazer um merge nas modificações do testing na branch main.

  $ git checkout main

  $ git merge testing

Rebase

- Outra maneira de juntar seu trabalho a um outro branch.

  $ git checkout experiment

  $ git rebase main

Delete branch

- Para excluir uma branch.

  $ git branch -D testing

Clonar branch

- Para excluir uma branch.

  $ git checkout -b testingfix origin/testingfix

POO

Exemplos de criação de classe

...continuando.

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 10: 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 src 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 11: 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
```

Padrões de Desenvolvimento de Software

Padrões de desenvolvimento de software

...continuando 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.
- Padrão Arquitetural MVC - Model-View-Controller (em português, Modelo-Visão-Controle ou também como Controlador de visualização de modelo) (MVC).

Padrões de criação

Propriedades dos padrões de criação:

- funcionam com base no modo como os objetos podem ser criados.
- isolam os detalhes da criação dos objetos.
- o código é independente do tipo do objeto a ser criado.

*obs: exemplo de padrão de criação é o padrão Singleton (solteiro).

Padrões estruturais

Propriedades dos padrões estruturais:

- eles determinam o design da estrutura de objetos e classes para que estes possam ser compostos e resultados mais amplos sejam alcançados.
- o foco está em simplificar a estrutura e identificar o relacionamento entre classes e objetos.
- estão centrados em herança e composição de classes.

*obs: exemplo de padrão estrutural é o padrão Adapter (adaptador).

Padrões comportamentais

Propriedades dos padrões comportamentais:

- estão preocupados com a interação entre os objetos e suas responsabilidades.
- os objetos devem ser capazes de interagir e, mesmo assim, devem ter baixo acomplamento.

*obs: exemplo de padrão comportamental é o padrão Observer (observador).

Padrão Arquitetural MVC

A ideia do MVC é ter um padrão de arquitetura cujo objetivo é **separar o projeto em três camadas independentes**, que são o modelo, a visão e o controlador.

- Essa **separação de camadas** ajuda na redução de acoplamento e promoção do aumento de coesão nas classes do projeto.
- Assim sendo, quando o modelo MVC é utilizado, pode facilitar a manutenção do código e sua reutilização em outros projetos.

*obs:

- **Baixo acoplamento**: é o grau em que uma classe conhece a outra.
- **Coesão**: uma classe com propósito bem definido (responsabilidade única) tem alta coesão, e isso é bom. Uma classe tem baixa coesão quando há propósitos que não pertencem apenas a ela, o que é ruim.

Padrão Arquitetural MVC

Explicando cada um dos objetos que o padrão MVC tem.

- Primeiramente o controlador (Controller), que interpreta as entradas enviadas ao aplicativo e mapeia essas ações em comandos que são enviados para o modelo (Model) e/ou para a janela de visualização (View) para efetuar a alteração apropriada.
- Por sua vez, o modelo (Model) gerencia um ou mais elementos de dados, responde a perguntas sobre o seu estado e responde a instruções para mudar de estado. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema a ser resolvido.
- Por fim, a visão (View) gerencia a saída de dados e é responsável por apresentar as informações para quem solicitou. A visão não sabe nada sobre o que a aplicação está atualmente fazendo, pois tudo que ela realmente faz é receber instruções do controle e informações do modelo e então devolve-las. A visão também se comunica de volta com o modelo e com o controlador para sinalizar seu estado.

Padrão Arquitetural MVC

...continuando com o MVC.

- Portanto, a principal ideia do modelo MVC é separar conceitos - e do código.
- O MVC é como a clássica programação orientada a objetos, ou seja, criar objetos que escondem as suas informações e como elas são manipuladas e então apresentadas em uma interface.
- Entre as diversas vantagens do padrão MVC estão a possibilidade de reescrita da Graphical User Interface (em português, Interface Gráfica do Utilizador) (GUI) ou do Controller sem alterar o modelo, reutilização da GUI para diferentes aplicações com pouco esforço, facilidade na manutenção e adição de recursos, reaproveitamento de código, facilidade na manutenção do "código limpo".

Padrão de projeto Singleton

O Singleton proporciona uma forma de ter um e somente um objeto de determinado tipo, além de disponibilizar um ponto de acesso global.

- Por isso, os Singletons são geralmente utilizados em casos como logging ou operações de banco de dados e muito outro cenários em que seja necessário que haja apenas uma instância disponível para toda a aplicação a fim de evitiar requisições conflitantes para o mesmo recurso.
- Uma maneira simples de implementar o Singleton é deixar o construtor privado e criar um método estático que faça a inicialização do objeto. Mas em Python a implementação é um pouco diferente, pois não há como criar construtor privado.

Vamos praticar

Padrão de projeto Singleton

```python
class Singleton(object):
    def __new__(cls, *args, **kwargs):
        if not hasattr(cls, 'instance'):
            cls.instance = super(Singleton, cls).__new__(cls)
            cls.instance._initialized = False
        return cls.instance

    def __init__(self, wrapped_class=None, *args, **kwargs):
        if wrapped_class and not self._initialized:
            self._initialized = True
            self.wrapped_class_instance = wrapped_class(*args, **kwargs)

    def update_instance(self, *args, **kwargs):
        if hasattr(self, 'wrapped_class_instance'):
            for key, value in kwargs.items():
                setattr(self.wrapped_class_instance, key, value)

    def __getattr__(self, name):
        return getattr(self.wrapped_class_instance, name)
```

Código 12: Padrão de projeto Singleton

*obs: na **class Singleton(object)**, o objeto é criado com o método __new__, mas antes disso é feita uma verificação para saber se o objeto já existe. O método *hasatt* (método especial de Python para saber se um objeto tem determinada propriedade) é usado para verificar se o objeto tem determinada propriedade *instance*, que confere se a classe já tem um objeto.

Pytest

```python
import pytest
from src import Singleton
from src import Produto

def testa_classe_singleton_se_duas_instancias_sao_iguais_com_sucesso():
  singleton = Singleton()
  assert id(singleton) == id(Singleton())
  # id() Retorna a identidade de um objeto.
  # Isso é garantido como único entre objetos existentes simultaneamente.
  # (CPython usa o endereço de memória do objeto.)

@pytest.mark.parametrize(('produto', 'nome', 'codigo', 'preco', 'quantidade',
                'produto2', 'nome2', 'codigo2', 'preco2', 'quantidade2'),
                [(Produto, 'Arroz', 1, 1, 1, Produto, 'Feijao', 1, 1, 1),
                 (Produto, 'Feijao', 1, 1, 1, Produto, 'Arroz', 1, 1, 1,)])
def testa_classe_singleton_se_duas_instancias_de_produto_sao_iguais_com_sucesso(
  produto, nome, codigo, preco, quantidade, produto2, nome2, codigo2, preco2, quantidade2):
  produto_inst = Singleton(produto, nome, codigo, preco, quantidade)
  produto_inst2 = Singleton(produto2, nome2, codigo2, preco2, quantidade2)
  assert produto_inst.obtem_nome() == produto_inst2.obtem_nome() \
    and produto_inst.obtem_nome() == 'Arroz' and produto_inst2.obtem_nome() == 'Arroz'

@pytest.mark.parametrize(('produto', 'nome', 'codigo', 'preco', 'quantidade',
                'produto2', 'nome2', 'codigo2', 'preco2', 'quantidade2'),
                [(Produto, 'Arroz', 1, 1, 1, Produto, 'Feijao', 1, 1, 1),
                 (Produto, 'Feijao', 1, 1, 1, Produto, 'Arroz', 1, 1, 1,)])
def testa_classe_singleton_atualizando_dados_na_instancia_com_sucesso(
  produto, nome, codigo, preco, quantidade, produto2, nome2, codigo2, preco2, quantidade2):
  produto_inst = Singleton(produto, nome, codigo, preco, quantidade)
  produto_inst2 = Singleton(produto2, nome2, codigo2, preco2, quantidade2)
  produto_inst.update_instance(produto=produto2, nome=nome2, codigo=codigo2,
                preco=preco2, quantidade=quantidade2)
  assert produto_inst.obtem_nome() == nome2 and produto_inst2.obtem_nome() == nome2
```

Código 13: Cobertura de testes da classe Singleton

Padrão de projeto Singleton Preguiçoso

Instanciação preguiçosa no padrão Singleton.

- A instanciação preguiçosa garante que o objeto seja criado quando realmente precisamos dele.
- Considere a instanciação preguiçosa como uma maneira de trabalhar como recursos reduzidos e criá-los somente quando houver necessidade.

Padrão de projeto Singleton Lazy

```python
class SingletonLazy:
    __instance = None
    @classmethod
    def get_instance(cls):
        if not cls.__instance:
            cls.__instance = SingletonLazy()
        return cls.__instance
    # Um classmethod é um método que recebe a classe como primeiro argumento
    # em vez da instância. Isso significa que você pode acessar atributos e
    # métodos de nível de classe a partir deste método. Um uso comum de
    # classmethod é para definir construtores alternativos ou implementar
    # método que gerencia instancias de classe.

    # Neste exemplo, get_instance é um método de classe que nos permite pegar
    # uma instância da classe por meio do atributo __instance.
    # O argumento cls refere-se a própria classe SingletonLazy, não a uma instância. Isso é útil porque se criarmos uma classe SingletonLazy, o método get_instance consulta
      automaticamente se há uma instância da classe, e não apenas um objeto SingletonLazy.
```

Código 14: Padrão de projeto Singleton Lazy

*obs: Um classmethod é um método que recebe a classe como primeiro argumento em vez da instância. Isso significa que você pode acessar atributos e métodos de nível de classe a partir deste método. Um uso comum de classmethod é para definir construtores alternativos. Para usar @classmethod, sempre passe cls como o primeiro parâmetro.

Para mais detalhes sobre decorados, clique aqui

Pytest

```python
from src import SingletonLazy

def testa_classe_singleton_lazy_se_duas_instancias_sao_iguais_com_sucesso():
  singleton = SingletonLazy().get_instance()
  assert id(singleton) == id(SingletonLazy().get_instance())
  # id() Retorna a identidade de um objeto.
  # Isso é garantido como único entre objetos existentes simultaneamente.
  # (CPython usa o endereço de memória do objeto.)
```

Código 15: Cobertura de testes da classe Singleton

POO

Exemplos de criação de classe

...continuando a partir da aula passada sobre Programação Orientada a Objetos (POO).

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 Produto, 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')
```

Script Python com passagem de parâmetros

Como usar os argumentos passados para um script python?

Arquivo principal.py com conteúdo:

```
import sys
for value in sys.argv:
print(value)
```

*obs: inicialize seu script assim: *python3 principal.py arg1 arg2 arg3*

Script Python com passagem de parâmetros

```python
import sys

def iniciar_app(argv) -> None:
    for value in argv:
        print(value)

if __name__ == '__main__':
    iniciar_app(sys.argv)
```

Código 16: Inicialização de script com parâmetros

*obs: inicialize seu script assim: *python3 principal.py arg1 arg2 arg3*

Pytest

```python
from src import iniciar_app
import pytest
from unittest.mock import patch
from io import StringIO
import sys

def test_parse_args():
  original_stdout = sys.stdout
  sys.stdout = StringIO()  # Redireciona stdout para capturar a saída
  test_argv = ['arg1', 'arg2', 'arg3'] # Argumentos de teste
  iniciar_app(test_argv)
  output = sys.stdout.getvalue() # Obtém a saída capturada
  sys.stdout = original_stdout  # Restaura o stdout original
  expected_output = 'arg1\narg2\narg3\n' # Verifica se a saída está correta
  assert output == expected_output

@pytest.mark.parametrize(
    "test_argv, expected_output",
    [(['arg1', 'arg2', 'arg3'], 'arg1\narg2\narg3\n'),(['foo', 'bar'], 'foo\nbar\n')]
)
def test_parse_args_with_patch(test_argv, expected_output):
  with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
    iniciar_app(test_argv) # Chama a função com os argumentos de teste
    output = mock_stdout.getvalue() # Obtém a saída capturada
    assert output == expected_output # Verifica se a saída está correta
```

Código 17: Cobertura de testes

Exercícios

Escreva o algoritmo e programa. Use POO para resolver.

- *Exercício 017*: Ler 4 números inteiros que correspondem ao número da conta do cliente, saldo, débito ou crédito. Os número serão passados na inicialização do script. Calcular e escrever o saldo atual (saldo atual = saldo - débito + crédito). Também verificar 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

...continuando Padrões de desenvolvimento de software.

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✗
  - Builder ✗
  - Factory Method ✗
  - Prototype ✗

Padrões de criação

Abstract Factory ✔

- fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
- Em outras palavras, ele permite que você crie objetos sem precisar saber exatamente qual classe concreta será instanciada.

Padrões de criação - Abstract Factory

Principais considerações:

1. **AbstractFactory**: Define uma interface para criar produtos abstratos. Normalmente, essa interface declara métodos para criar cada tipo de produto que a fábrica pode produzir.
2. **ConcreteFactory**: Implementa a interface da fábrica abstrata e produz instâncias concretas dos produtos.
3. **AbstractProduct**: Declara uma interface para um tipo de produto. Cada produto específico deve implementar essa interface.
4. **ConcreteProduct**: Implementa a interface do produto abstrato. Cada fábrica concreta criará instâncias desse tipo de produto.
5. **Client**: Usa apenas as interfaces definidas pela fábrica abstrata e pelos produtos abstratos para trabalhar com os objetos. O cliente não precisa saber quais classes concretas estão sendo usadas.

Padrões de criação - Abstract Factory

Tem por objetivo permitir a criação de famílias de objetos relacionados sem depender de suas classes concretas. Em Python, esse padrão é frequentemente usado para fornecer uma *interface* para criar objetos em uma **superclasse**, enquanto permite que **subclasses** alterem o tipo de objetos que serão criados.

1. Define uma interface para criar objetos, mas não implementa os métodos de criação.
2. Subclasses específicas (concretas) implementam a criação dos objetos, fornecendo diferentes implementações de uma família de objetos relacionados.

Padrão de projeto Abstract Factory

```python
import abc

class CadeiraAbstrata(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def sentar_em(self):
        pass
```

Código 18: Código da Classe Cadeira Abstrata

```python
import abc

class SofaAbstrato(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def sentar_em(self):
        pass
```

Código 19: Código da Classe Sofa Abstrato

Padrão de projeto Abstract Factory

```python
from src import CadeiraAbstrata

class CadeiraClassicaConcreta(CadeiraAbstrata):
    def sentar_em(self):
        return "Sentado em uma cadeira classica"
```

Código 20: Código da Classe Cadeira Concreta

```python
from src import SofaAbstrato

class SofaClassicoConcreta(SofaAbstrato):
    def sentar_em(self):
        return "Sentado em um sofa classico"
```

Código 21: Código da Classe Sofa Concreta

Padrão de projeto Abstract Factory

```python
from src import CadeiraAbstrata

class CadeiraModernaConcreta(CadeiraAbstrata):
    def sentar_em(self):
        return "Sentado em uma cadeira moderna"
```

Código 22: Código da Classe Cadeira Moderna

```python
from src import SofaAbstrato

class SofaModernoConcreta(SofaAbstrato):
    def sentar_em(self):
        return "Sentado em um sofa moderno"
```

Código 23: Código da Classe Sofa Moderna

Padrão de projeto Abstract Factory

```python
import abc
from src import CadeiraAbstrata, SofaAbstrato

class MoveisFactoryAbstrato(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def criar_cadeira(self) -> CadeiraAbstrata:
        pass

    @abc.abstractmethod
    def criar_sofa(self) -> SofaAbstrato:
        pass
```

Código 24: Código da Classe Moveis Factory Abstrato

Padrão de projeto Abstract Factory

```python
from src import CadeiraAbstrata, SofaAbstrato, MoveisFactoryAbstrato, SofaClassicoConcreta, CadeiraClassicaConcreta

class MoveisClassicosFactoryConcreto(MoveisFactoryAbstrato):
    def criar_cadeira(self) -> CadeiraAbstrata:
        return CadeiraClassicaConcreta()

    def criar_sofa(self) -> SofaAbstrato:
        return SofaClassicoConcreta()
```

Código 25: Código da Classe Cadeira Classica Factory Concretra

```python
from src import CadeiraAbstrata, SofaAbstrato, MoveisFactoryAbstrato, SofaModernoConcreta, CadeiraModernaConcreta

class MoveisModernosFactoryConcreto(MoveisFactoryAbstrato):
    def criar_cadeira(self) -> CadeiraAbstrata:
        return CadeiraModernaConcreta()

    def criar_sofa(self) -> SofaAbstrato:
        return SofaModernoConcreta()
```

Código 26: Código da Classe Sofa Classica Factory Concretra

Padrão de projeto Abstract Factory

```python
from src import MoveisFactoryAbstrato

# Cliente usando a Abstract Factory
def codigo_cliente(factory: MoveisFactoryAbstrato):
    cadeira = factory.criar_cadeira()
    sofa = factory.criar_sofa()

    print(cadeira.sentar_em())
    print(sofa.sentar_em())
```

Código 27: Código da função código do cliente

Pytest

```python
import pytest
from unittest.mock import patch
from src import MoveisModernosFactoryConcreto, MoveisClassicosFactoryConcreto, codigo_cliente

@pytest.mark.parametrize("valor_esperado", [(['Sentado em uma cadeira moderna', 'Sentado em um sofa moderno'])])
def teste_codigo_cliente_para_usar_moveis_modernos_com_sucesso(valor_esperado):
    with patch('builtins.print') as mock_print:
        moderna_factory = MoveisModernosFactoryConcreto()
        codigo_cliente(moderna_factory)
        mock_print.assert_any_call(valor_esperado[0])
        mock_print.assert_any_call(valor_esperado[1])
        assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(['Sentado em uma cadeira classica', 'Sentado em um sofa classico'])])
def teste_codigo_cliente_para_usar_moveis_classicos_com_sucesso(valor_esperado):
    with patch('builtins.print') as mock_print:
        moderna_factory = MoveisClassicosFactoryConcreto()
        codigo_cliente(moderna_factory)
        mock_print.assert_any_call(valor_esperado[0])
        mock_print.assert_any_call(valor_esperado[1])
        assert mock_print.call_count == len(valor_esperado)
```

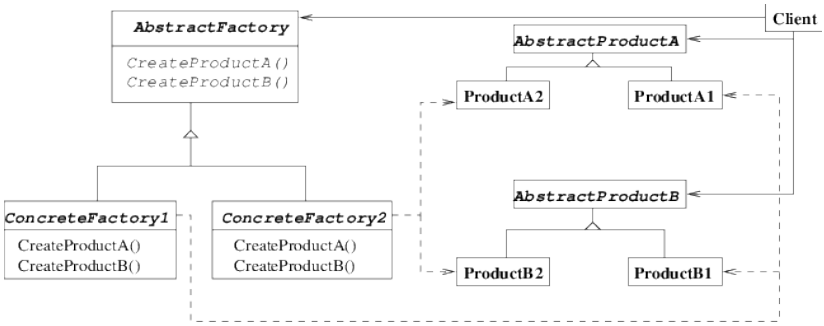
Código 28: Cobertura de testes da função codigo cliente

Padrões de criação - Abstract Factory

Resumo

1. O padrão Abstract Factory permite a criação de famílias de objetos relacionados (por exemplo, móveis modernos ou classicos) sem especificar as classes concretas que os compõem.
2. Isso promove a flexibilidade e extensibilidade no código, já que novas famílias de objetos podem ser introduzidas sem modificar o código existente.

Abstract Factory



Exercícios

Escreva o algoritmo e programa. Use POO para resolver.

- *Exercício 029*: Na loja de seu Zé, são vendidos produto novos e usados. No produto escreva 'produto novo' se o produto for novo e 'produto usado' caso seja um produto usado. A classe produto deve possuir um método para escrever o estado do produto (novo ou usado). Aplique os conceitos aprendidos sobre o padrão Abstract Factory.

POO

Exemplos de criação de classe

...continuando a partir da aula passada sobre POO.

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 src.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 29: 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 src 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 30: 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-backer
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%
```

Exercícios

Escreva o algoritmo e programa. Use Herança de POO para resolver.

- *Exercício 018*: Ler 3 números inteiros que correspondem a (1) quantidade atual em estoque, (2) quantidade máxima em estoque e (3) quantidade mínima em estoque de um produto. Calcular e escrever a quantidade média ((quantidade média = quantidade máxima + quantidade mínima)/2). Se a quantidade em estoque for maior ou igual a quantidade média escrever a mensagem 'Não efetuar compra', senão escrever a mensagem 'Efetuar compra'.

Padrões de Desenvolvimento de Software

Padrões de desenvolvimento de software

...continuando Padrões de desenvolvimento de software.

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✔
  - Factory Method ✗
  - Builder ✗
  - Prototype ✗

Padrões de criação

Factory Method ✔

- Padrão que define um método para criar um objeto em uma classe-mãe, mas permite que subclasses modifiquem o tipo de objeto que será criado.
- Ele é ideal quando você tem uma superclasse que faz algo, mas quer que subclasses específicas decidam como criar esses objetos.

Padrões de criação

Exemplo

- Imagine que você tenha uma classe Documento que é responsável por abrir arquivos.
- Dependendo do tipo de documento (PDF, Word), você quer abrir o documento de maneiras diferentes.
- O **Factory Method** permite que cada tipo de documento implemente seu próprio método de abertura.

Padrão de projeto Factory Method

```python
import abc

class DocumentoAbstrata(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def abrir(self):
        pass
```

Código 31: Código da Classe Documento - codigo_016_documento_abstrata.py Abstrata

```python
from src import DocumentoAbstrata

class DocumentoPdfConcreta(DocumentoAbstrata):
    def abrir(self):
        return "Abrindo um documento PDF"
```

Código 32: Código da Classe Documento Pdf - codigo_016_documento_pdf_concreta.py Concreta

```python
from src import DocumentoAbstrata

class DocumentoMsWordConcreta(DocumentoAbstrata):
    def abrir(self):
        return "Abrindo um documento Microsoft Word"
```

Código 33: Código da Classe Documento Ms Word - codigo_016_documento_msword_concreta.py Concreta

Padrão de projeto Factory Method

```python
import sys
import os
# Adiciona o caminho do diretório 'atual' ao sys.path.
# Isso resolver problemas com imports de módulos que estão no mesmo diretório.
sys.path.append(os.path.join(os.path.dirname(__file__), '..', ''))
from src import DocumentoPdfConcreta
from src import DocumentoMsWordConcreta

# Cliente usando a Factory Method
def codigo_cliente_factory_method(tipo_documento: str):

    if(tipo_documento == "word" or tipo_documento == "pdf"):

        if tipo_documento == "word":
            documento = DocumentoMsWordConcreta()

        elif tipo_documento == "pdf":
            documento = DocumentoPdfConcreta()

        print(documento.abrir())
    else:

        print("Tipo de documento desconhecido")
```

Código 34: Código da função codigo_cliente_factory_method

*obs: Linhas de 1 a 6 são para ajustar a leitura dos arquivos para o Phyton, que estão no mesmo diretório.

Padrão de projeto Factory Method

```python
import pytest
from unittest.mock import patch
from src import codigo_cliente_factory_method

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento PDF"])])
def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("pdf")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento Microsoft Word"])])
def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("word")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Tipo de documento desconhecido"])])
def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("outro")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)
```

Código 35: Testes do Código do cliente para utilizar a Factory Method - test_codigo_016_cliente_factory_method.py

Padrão de projeto Factory Method

```python
from src import DocumentoMsWordConcreta, DocumentoPdfConcreta

class DocumentoFactoryMethodConcreta:

    def __init__(self, tipo_documento: str):
        if tipo_documento == "word":
            self.document = DocumentoMsWordConcreta()
        elif tipo_documento == "pdf":
            self.document = DocumentoPdfConcreta()

    def abrir(self):
        return self.document.abrir()
```

Código 36: Código da Classe Documento Factory Method Concreta - codigo_016_documento_factory_method_concreta.py

*obs: quaias as vantagens e desvantagens desta "segunda" implementação? que utilizou a classe **DocumentoFactoryMethodConcreta**.

Padrão de projeto Factory Method

```python
from src import DocumentoFactoryMethodConcreta

# Cliente usando a Factory Method
def codigo_cliente_factory_method2(tipo_documento: str):

    if(tipo_documento == "word" or tipo_documento == "pdf"):

        documento = DocumentoFactoryMethodConcreta(tipo_documento)

        print(documento.abrir())

    else:

        print("Tipo de documento desconhecido")
```

Código 37: Código do cliente para utilizar a Factory Method - codigo_016_codigo_cliente_factory_method2.py

Padrão de projeto Factory Method

```python
import pytest
from unittest.mock import patch
from src import codigo_cliente_factory_method2

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento PDF"])])
def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("pdf")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento Microsoft Word"])])
def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("word")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Tipo de documento desconhecido"])])
def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("outro")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)
```

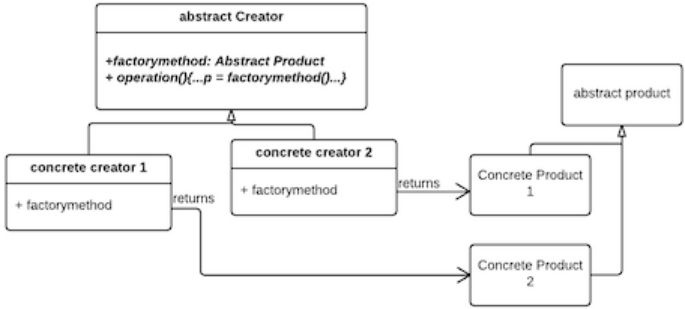
Código 38: Testes do Código do cliente para utilizar a Factory Method

Padrões de criação

Diferenças Fundamentais entre Abstract Factory e Factory Method?

- **Escopo de Aplicação**
  - Abstract Factory: Focado na criação de famílias de objetos relacionados.
  - Factory Method: Focado na criação de um único objeto, delegando a subclasses a decisão de qual tipo de objeto criar.
- **Complexidade**
  - Abstract Factory: Envolve várias classes e métodos para criar diferentes objetos relacionados.
  - Factory Method: Mais simples, envolvendo um único método de criação.
- **Flexibilidade**
  - Abstract Factory: Útil para criar produtos consistentes entre si.
  - Factory Method: Útil quando a criação de objetos pode variar em subclasses.

Abstract Factory



POO

Programação Orientada a Objetos

...continuando a partir da aula passada sobre POO.

Programação Orientada a Objetos

...relembrando a última aula:

- 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.

Exercícios

Escreva o algoritmo e programa. Use Herança de POO para resolver.

- *Exercício 018*: Ler 3 números inteiros que correspondem a (1) quantidade atual em estoque, (2) quantidade máxima em estoque e (3) quantidade mínima em estoque de um produto. Calcular e escrever a quantidade média ((quantidade média = quantidade máxima + quantidade mínima)/2). Se a quantidade em estoque for maior ou igual a quantidade média escrever a mensagem 'Não efetuar compra', senão escrever a mensagem 'Efetuar compra'.

Interface de classe

Uma interface define um conjunto de métodos que uma classe deve implementar. Ela atua como um contrato.

- Em Python, não há uma palavra-chave específica para definir **interfaces**, mas podemos usar **classes abstratas** para criar **interfaces**.

Classes Abstratas

Uma classe abstrata é uma classe que não pode ser instanciada diretamente. Ela serve como um modelo para outras classes.

- Podemos criar uma classe abstrata com o **módulo abc** (Abstract Base Classes) em Python.

https://docs.python.org/pt-br/3/library/abc.html

Interface com Classes Abstratas - Caso com abc

```python
import abc
class ProdutoAbstrato(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
        pass
    @abc.abstractmethod
    def obtem_nome(self)-> str:
        pass
    @abc.abstractmethod
    def obtem_codigo(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_preco(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_serie(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_lote(self)-> int:
        pass
    @abc.abstractmethod
    def altera_preco(self, novo_preco) -> bool:
        pass
    @abc.abstractmethod
    def altera_quantidade(self, novo_pedido: int) -> bool:
        pass
    @abc.abstractmethod
    def obtem_produto(self) -> str:
        pass
```

Classe Produto Abstrata - codigo_012_classe_produto_abstrato.py

*obs: **pass** é usada como um espaço reservado quando o código é necessário, mas nenhuma ação específica é exigida. Use **pass** como um espaço reservado para posteriormente implementação.

Interface com Classes Abstratas - Caso com abc

```python
from src import ProdutoAbstrato

class ProdutoConcreto(ProdutoAbstrato):
    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)
```

Pytest

```python
from src import ProdutoConcreto
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def testa_classe_produto_concreto_geral_com_sucesso(nome, codigo, preco, quantidade):
        produto = ProdutoConcreto(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
        assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo, preco, quantidade)
        assert produto.altera_preco(2) == True
        assert produto.altera_preco(1) == False
        assert produto.altera_quantidade(2) == False
        assert produto.altera_quantidade(1) == True
```

Cobertura de testes da classe Produto Concreto - test_codigo_012_classe_produto_concreto.py

Testes Unitários

```
---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                 Stmts   Miss  Cover   Missing
----------------------------------------------------------------------------------
src/__init__.py                                         15      0   100%
src/codigo_001_somar.py                                  2      0   100%
src/codigo_002_lambda_somar.py                           1      0   100%
src/codigo_003_classe_produto.py                        31      0   100%
src/codigo_004_classe_produto_critico.py                19      0   100%
src/codigo_005_classe_produto_perecivel.py               7      0   100%
src/codigo_006_classe_veiculo_com_exception.py          18      0   100%
src/codigo_008_input.py                                 14      0   100%
src/codigo_009_print.py                                  9      0   100%
src/codigo_010_classe_singleton.py                      16      0   100%
src/codigo_011_classe_singleton_lazy.py                  7      0   100%
src/codigo_012_classe_produto_abstrato.py                2      0   100%
src/codigo_012_classe_produto_concreto.py               32      0   100%
src/codigo_013_classe_produto_abstrato_excecao.py       10      0   100%
src/codigo_013_classe_produto_concreto_excecao.py       32      0   100%
----------------------------------------------------------------------------------
TOTAL                                                  215      0   100%
Coverage HTML written to dir htmlcov

Required test coverage of 100% reached. Total coverage: 100.00%
```

*obs: *pytest –maxfail=1 –cov-fail-under=100 –disable-warnings –cov=./src –cov-report=html –cov-report=term-missing*

Interface com Classes Abstratas - Caso com Exceção

```python
class ProdutoAbstratoExcecao:

    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
        raise NotImplementedError()
    def obtem_nome(self)-> str:
        raise NotImplementedError()
    def obtem_codigo(self)-> int:
        raise NotImplementedError()
    def obtem_preco(self)-> int:
        raise NotImplementedError()
    def obtem_serie(self)-> int:
        raise NotImplementedError()
    def obtem_lote(self)-> int:
        raise NotImplementedError()
    def altera_preco(self, novo_preco) -> bool:
        raise NotImplementedError()
    def altera_quantidade(self, novo_pedido: int) -> bool:
        raise NotImplementedError()
    def obtem_produto(self) -> str:
        raise NotImplementedError()
```

Classe Produto Abstrato Exceção - codigo_013_classe_produto_abstrato_excecao.py

Interface com Classes Abstratas - Caso com Exceção

```python
from src import ProdutoAbstratoExcecao

class ProdutoConcretoExcecao(ProdutoAbstratoExcecao):
    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)
```

Classe Produto Concreto Exceção - codigo_013_classe_produto_abstrato_excecao.py

Testes Unitários

```python
from src import ProdutoConcretoExcecao
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def testa_classe_produto_concreto_excecao_geral_com_sucesso(nome, codigo, preco, quantidade):
        produto = ProdutoConcretoExcecao(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
        assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo, preco, quantidade)
        assert produto.altera_preco(2) == True
        assert produto.altera_preco(1) == False
        assert produto.altera_quantidade(2) == False
        assert produto.altera_quantidade(1) == True
```

Cobertura de testes da classe Produto Concreto Exceção - test_codigo_013_classe_produto_concreto_excecao.py

Padrões de Desenvolvimento de Software

Padrões de desenvolvimento de software

...continuando Padrões de desenvolvimento de software.

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✔
  - Factory Method ✔
  - Builder ✗
  - Prototype ✗

Padrões de criação - Builder

Builder (Construtor - 'de construir coisas') ✔

- Padrão de criação que se concentra em como construir objetos complexos de maneira controlada e eficiente.
- **Ele separa a construção de um objeto da sua representação final, permitindo a criação de diferentes representações ou configurações do mesmo objeto.**
- Esse padrão é especialmente útil quando um objeto precisa ser construído passo a passo, ou quando o processo de construção é muito complexo.

Padrões de criação - Builder

Exemplo

1. **Classe ProdutoBase** - Será o objeto complexo que queremos construir. Ele pode ter várias partes diferentes que são configuradas durante o processo de construção.
2. **Classe BuilderAbstrata** - É uma interface ou classe abstrata que define os métodos para criar as diferentes partes do Produto.
3. **Classe BuilderConcreta** - Implementa a interface do Builder e constrói as partes específicas do Produto.
4. **Classe Diretor** - define a ordem de construção das partes do Produto. Ele usa o Builder para construir o Produto passo a passo.
5. **Código do cliente para utilizar o Padrão Builder** - Quem usa o Diretor e o Builder para construir os objetos complexos.

Padrões de criação - Builder

Resumo

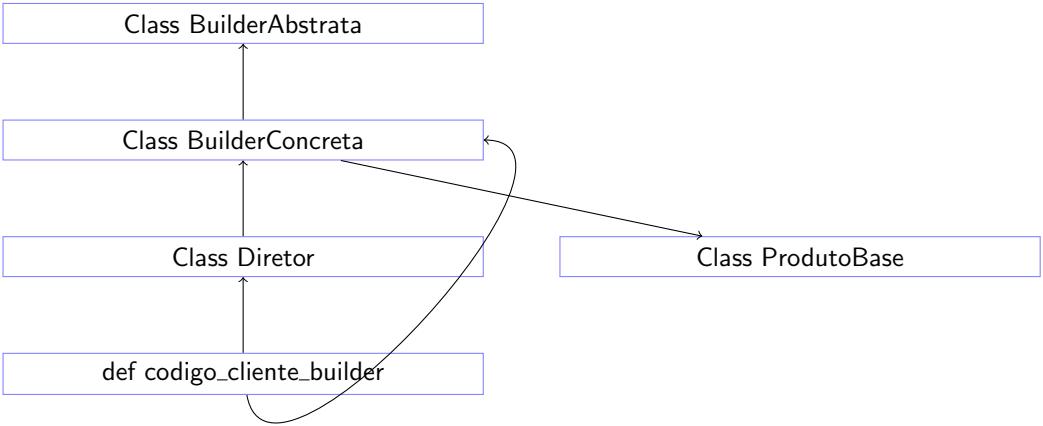
- Produto: O objeto que está sendo construído.
- Builder: Interface que define os métodos de construção.
- BuilderConcreta: Implementa os métodos do Builder e constrói as partes do Produto.
- Diretor: Controla o processo de construção, definindo a ordem em que as partes são construídas.

Padrões de criação - Builder

Aplicação

- O padrão Builder é útil quando queremos construir objetos que exigem uma criação passo a passo, ou quando diferentes representações do mesmo objeto podem ser necessárias.

Padrões de criação - Builder



Padrões de criação - Builder

Class ProdutoBase	Possui →	Lógica do produto.
Class BuilderAbstrata	Possui →	Interface para definir a construção do produto.
Class BuilderConcreta	Possui →	Implementa a lógica para definir a construção do produto.
Class DiretorConcreta	Possui →	Implementa a lógica para definir a ordem de construção do produto.
def codigo_cliente_builder	Possui →	Implementa o código do cliente para obter o produto final.

Padrão de projeto - Builder

```python
from typing import Any

class ProdutoBase():
    def __init__(self)-> None:
        self._partes = []

    def adicionar_parte(self, parte: Any) -> None:
        self._partes.append(parte)

    def __str__(self):
        return f"Partes do produto: {', '.join(self._partes)}"
```

Código da Classe Produto Base - codigo_017_produto_base.py

Padrão de projeto - Builder

```python
import abc
from .codigo_017_produto_base import ProdutoBase

class BuilderAbstrato(metaclass=abc.ABCMeta):

    @abc.abstractmethod
    def limpar(self) -> None:
        pass

    @abc.abstractmethod
    def produto(self)-> ProdutoBase:
        pass

    @abc.abstractmethod
    def construir_parte_a(self)-> None:
        pass

    @abc.abstractmethod
    def construir_parte_b(self)-> None:
        pass

    @abc.abstractmethod
    def construir_parte_c(self)-> None:
        pass
```

Código da Classe Builder Abtratato - codigo_017_builder_abstrato.py

Padrão de projeto - Builder

```python
from .codigo_017_builder_abstrato import BuilderAbstrato
from .codigo_017_produto_base import ProdutoBase

class BuilderConcreta(BuilderAbstrato):

    def __init__(self):
        self.limpar()

    def limpar(self) -> None:
        self.__produtoBase = ProdutoBase()

    @property
    def produto(self)-> ProdutoBase:
        produto = self.__produtoBase
        self.limpar()
        return produto

    def construir_parte_a(self) -> None:
        self.__produtoBase.adicionar_parte("Parte a")

    def construir_parte_b(self)-> None:
        self.__produtoBase.adicionar_parte("Parte b")

    def construir_parte_c(self)-> None:
        self.__produtoBase.adicionar_parte("Parte c")
```

Código da Classe Builder Concreta - codigo_017_builder_concreta.py

Padrão de projeto - Builder

```python
from .codigo_017_builder_abstrato import BuilderAbstrato

class Diretor:
    def __init__(self)-> None:
        self.__construir = None

    @property
    def construir(self) -> BuilderAbstrato:
        return self.__construir

    @construir.setter
    def construir(self, construir: BuilderAbstrato) -> None:
        self.__construir = construir

    def construir_produto_simples(self) -> None:
        self.__construir.construir_parte_c()

    def construir_produto_sofisticado(self) -> None:
        self.__construir.construir_parte_a()
        self.__construir.construir_parte_b()
        self.__construir.construir_parte_c()
```

Código da Classe Diretor Concreta - codigo_017_diretor_base.py

Padrão de projeto - Builder

```python
# trecho de código para que seja possível você inicializar direto este script
# com base nos pacotes.
# início
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))) # volta 2 pastas
# fim
from padroesdeprojetos.criacao.codigo_017_builder_concreta import BuilderConcreta
from padroesdeprojetos.criacao.codigo_017_diretor_base import Diretor

def codigo_cliente_builder():
    diretor = Diretor()
    diretor.construir = BuilderConcreta()

    diretor.construir_produto_simples()
    print(f"{diretor.construir.produto}")

    diretor.construir_produto_sofisticado()
    print(f"{diretor.construir.produto}")

if __name__ == "__main__":
    codigo_cliente_builder()
```

Código da Função Código do cliente - codigo_017_codigo_cliente_builder.py

Pytest

```python
import pytest
from unittest.mock import patch
from src.padroesdeprojetos.criacao import codigo_cliente_builder

@pytest.mark.parametrize("valor_esperado", [("Partes do produto: Parte c",
↪  "Partes do produto: Parte a, Parte b, Parte c")])
def teste_codigo_cliente_builder_com_sucesso(valor_esperado):
        with patch('builtins.print') as mock_print:
                codigo_cliente_builder()
                mock_print.assert_any_call(valor_esperado[0])
                mock_print.assert_any_call(valor_esperado[1])
                assert mock_print.call_count == len(valor_esperado)
```

Código do teste para o Código do cliente - test_codigo_017_codigo_cliente_builder.py

Padrões de desenvolvimento de software

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✔
  - Factory Method ✔
  - Builder ✔
  - Prototype ✗

Padrões de criação - Prototype

Prototype (Protótipo) ✔

- Padrão criacional que permite a criação de novos objetos copiando ou clonando instâncias existentes, em vez de criar novas instâncias do zero.
- Isso é útil quando o custo de criação de um novo objeto é alto ou quando você deseja evitar a complexidade de inicializar um objeto em seu estado inicial.

Padrões de criação - Prototype

Exemplo

1. **Classe PrototypeBase** - Uma interface ou classe abstrata que define o método clone(), que é responsável por criar uma cópia do objeto.
2. **Classe PrototypeConcreta1 e PrototypeConcreta2** - Classes concretas que implementam a interface Prototype e sobrescrevem o método clone() para retornar uma cópia do objeto.
3. **Código do cliente para utilizar o Padrão Prototype** - O local que utiliza o método clone() para criar novos objetos a partir de protótipos.

Padrões de criação - Prototype

Resumo

- Prototype: Define a interface clone() que as subclasses precisam implementar.
- PrototypeConcreta1 e PrototypeConcreta2: Implementam a interface Prototype e fornecem a implementação do método clone() que usa copy.deepcopy() para criar uma cópia profunda do objeto.
- Cliente: O cliente cria novos objetos clonando os protótipos.

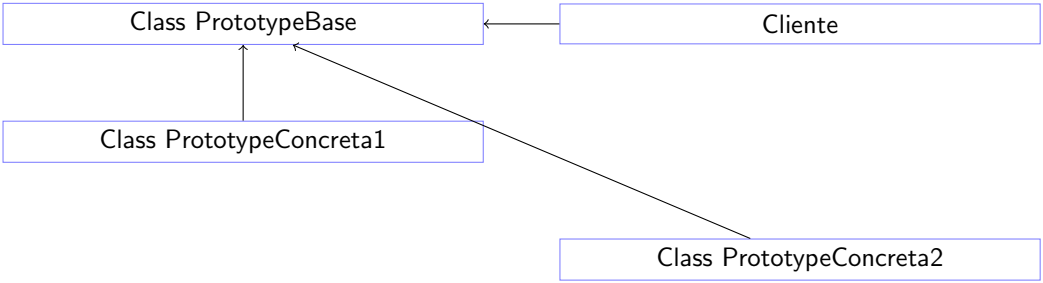
Padrões de criação - Prototype

Aplicação

- Quando o custo de criação de um novo objeto é muito caro ou complexo.
- Quando você deseja evitar a duplicação do estado de configuração dos objetos.
- Quando você precisa de uma variedade de objetos semelhantes.

O padrão Prototype é particularmente útil em cenários onde os objetos têm um estado inicial complexo ou quando você precisa criar cópias de objetos com uma configuração específica que não pode ser facilmente reproduzida.

Padrões de criação - Prototype



Padrões de criação - Prototype

Class PrototypeBase	Possui →	Interface para definir o clone de objeto.
Class PrototypeConcreta1	Possui →	Implementa a lógica para definir o clone do objeto PrototypeConcreta1.
Class PrototypeConcreta2	Possui →	Implementa a lógica para definir o clone do objeto PrototypeConcreta2.
def codigo_cliente_prototype	Possui →	Implementa o código do cliente para obter o clone.

Padrão de projeto - Prototype

```python
class PrototypeBase():
    def clone(self):
        """Deve retornar uma copia do objeto."""
        pass
```

Código da Classe Prototype Base - codigo_018_prototype_base.py

Padrão de projeto - Prototype

```python
from .codigo_018_prototype_base import PrototypeBase
import copy

class PrototypeConcreta1(PrototypeBase):
    def __init__(self, value):
        self.value = value

    def clone(self):
        # Usa a função deepcopy para criar uma cópia do objeto
        # Leia mais em https://docs.python.org/pt-br/3/library/copy.html
        return copy.deepcopy(self)

    def __str__(self):
        return f"PrototypeConcreta1 com valor: {self.value}"
```

Código da Classe Prototype Concreta 1 - codigo_018_prototype_concreta1.py

https://docs.python.org/pt-br/3/library/copy.html

Padrão de projeto - Prototype

```python
from .codigo_018_prototype_base import PrototypeBase
import copy

class PrototypeConcreta2(PrototypeBase):
    def __init__(self, value):
        self.value = value

    def clone(self):
        # Usa a função deepcopy para criar uma cópia do objeto
        # Leia mais em https://docs.python.org/pt-br/3/library/copy.html
        return copy.deepcopy(self)

    def __str__(self):
        return f"PrototypeConcreta2 com valor: {self.value}"
```

Código da Classe Prototype Concreta 2 - codigo_018_prototype_concreta2.py

https://docs.python.org/pt-br/3/library/copy.html

Padrão de projeto - Prototype

```python
if __name__ == "__main__":
    import sys
    import os
    sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))) # volta 2 pastas
    from padroesdeprojetos.criacao.codigo_018_prototype_concreta1 import PrototypeConcreta1
    from padroesdeprojetos.criacao.codigo_018_prototype_concreta2 import PrototypeConcreta2
else:
    from .codigo_018_prototype_concreta1 import PrototypeConcreta1
    from .codigo_018_prototype_concreta2 import PrototypeConcreta2

def codigo_cliente_prototype():
    # Criando os protótipos
    prototype1 = PrototypeConcreta1("Valor 1")
    prototype2 = PrototypeConcreta2("Valor 2")

    # Clonando os protótipos
    clone1 = prototype1.clone()
    clone2 = prototype2.clone()

    # Exibindo os objetos clonados
    print(f"{clone1}")  # PrototypeConcreta1 com Valor: Valor 1
    print(f"{clone2}")  # PrototypeConcreta2 com Valor: Valor 2

if __name__ == "__main__":
    codigo_cliente_prototype()
```

Código da Função Código do cliente - codigo_018_codigo_cliente_prototype.py

Testes Unitários

```python
import pytest
from unittest.mock import patch
from src.padroesdeprojetos.criacao import codigo_cliente_prototype

@pytest.mark.parametrize("valor_esperado", [(["PrototypeConcreta1 com valor: Valor 1", "PrototypeConcreta2 com valor: Valor 2"])])
def teste_codigo_cliente_prototype_com_sucesso(valor_esperado):
        with patch('builtins.print') as mock_print:
                codigo_cliente_prototype()
                mock_print.assert_any_call(valor_esperado[0])
                mock_print.assert_any_call(valor_esperado[1])
                assert mock_print.call_count == len(valor_esperado)
```

Código do teste para o Código do cliente - test_codigo_018_codigo_cliente_prototype.py

POO

Exemplos de criação de classe

...continuando a partir da aula passada sobre POO.

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.python.org/pt-br/3/whatsnew/2.6.html#pep-3110-exception-handling-changes

https://docs.python.org/pt-br/3/reference/compound_stmts.html#except-clause

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 39: Tratamento de exceções

Pytest

```python
from src 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 40: Cobertura de testes da classe Veiculo

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 src.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 41: 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 src 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 42: 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.
```

Soft Skills

Soft Skills

São habilidades subjetivas.

- Tem muita relação com a inteligência emocional.
- Muitas vezes são difíceis de serem identificadas e definidas.

Soft Skills

- Para entender a importância das soft skills é preciso deixar claro a sua definição.
- Porém não é fácil definir soft skills, pois essa definição pode variar com o contexto, por exemplo: uma competência que pode ser considerada soft skills numa área e pode ser considerada hard skill em outras áreas.

Soft Skills

Algumas boas definições.

- Competências comportamentais referentes a traços de personalidade e atitudes que guiam o comportamento do indivíduo.
- Combinação de qualidades pessoais, habilidades interpessoais, habilidades e conhecimentos adicionais que ajudam a obter uma boa performance profissional.
- Conjunto de competências e habilidades pessoais que descrevem as interações sociais, principalmente no ambiente de trabalho, além disso ditam a atitude de cada um e sua a compatibilidade com os outros.

Soft Skills

Soft skills encontradas no Computing Curricula de 2020 (p.50). Veja aqui.

- Pensamento Analítico e Crítico.
- Colaboração e Trabalho em Equipe.
- Perspectivas Éticas e Interculturais.
- Priorização e gerenciamento de multitarefas.
- Comunicação Oral e Apresentação.
- Resolução de problemas.
- Proatividade.
- Comunicação escrita.
- Gerenciamento de tempo.
- Autodidatismo.

Soft Skills

- Boas habilidades sociais têm um impacto positivo no ambiente de trabalho e na carreira dos profissionais.
- Entre as soft skills importantes, destacam-se a comunicação e a criatividade.
- A proficiência na comunicação oral e escrita é considerada um requisito mínimo para alguns cargos e carreiras.
- Além disso, a criatividade, muitas vezes vista como uma habilidade exclusiva de artistas, é essencial para encontrar soluções inovadoras para problemas.

Soft Skills

- Para ter sucesso no mercado de trabalho, os candidatos a emprego precisam ter uma vantagem competitiva sobre os outros candidatos com qualificações técnicas semelhantes, é aí que as soft skills se encaixam.
- Os empregadores preferem candidatos que estejam prontos para trabalhar, se uma pessoa (potencial empregado) tiver que ser treinado, em relação a soft skills, antes de se tornar produtivo para empresa, isso lhe dará uma desvantagem competitiva.

Soft Skills

- Durante a entrevista de emprego boas habilidades de comunicação são muito importantes e podem camuflar a falta de habilidades técnicas.
- Boas habilidades técnicas não são necessariamente suficientes para que um funcionário receba uma promoção, os empregadores preferem promover funcionários com soft skills bem desenvolvidas.

Soft Skills

Avaliação de empregadores de quais soft skills são essenciais para ter uma carreira bem sucedida

COMPETENCIES	WEIGHTED AVERAGE RATING*
Critical Thinking/Problem Solving	4.65
Teamwork/Collaboration	4.57
Professionalism/Work Ethic	4.48
Oral/Written Communications	4.35
Leadership	3.63
Information Technology Application	3.61
Career Management	3.32
Global/Multi-cultural Fluency	2.83

*5-point scale, where 1=Not essential, 2=Not very essential, 3=Somewhat essential, 4=Essential, 5=Absolutely essential.

Fonte: Job Outlook 2020 Survey Leia mais aqui

Backend

Python com Django

Prof. Bruno Iran Ferreira Maciel

BRUNO

Aug 2024

Exercícios 1

Exercícios 2

Exercícios 3

Exercícios 4

Referências

Referências 1

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