

**Backend**  
**Python com Django**  
**Prof. Bruno Iran Ferreira Maciel**



**BRUNO**

Aug 2024

---

# Estrutura de Tópicos das Aulas

1	<a href="#">Apresentação e boas-vindas</a>	14	11	<a href="#">POO</a>	131
2	<a href="#">Lógica de Prog. e Padrões de Des. de Software</a>	20	12	<a href="#">Padrões de Desenvolvimento de Software</a>	138
3	<a href="#">Lógica de Programação</a>	22	13	<a href="#">POO</a>	153
4	<a href="#">Padrões de Desenvolvimento de Software</a>	27	14	<a href="#">Padrões de Desenvolvimento de Software</a>	165
5	<a href="#">Lógica de Programação</a>	33	15	<a href="#">POO</a>	180
6	<a href="#">Padrões de Desenvolvimento de Software</a>	62	16	<a href="#">Padrões de Desenvolvimento de Software</a>	209
7	<a href="#">POO</a>	78	17	<a href="#">POO</a>	221
8	<a href="#">Padrões de Desenvolvimento de Software</a>	83	18	<a href="#">Padrões de Desenvolvimento de Software</a>	233
9	<a href="#">POO</a>	85	19	<a href="#">POO</a>	259
10	<a href="#">Git</a>	103	20	<a href="#">Soft Skills</a>	273

# 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	Impensado - 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 software 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 Programação.
- 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

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

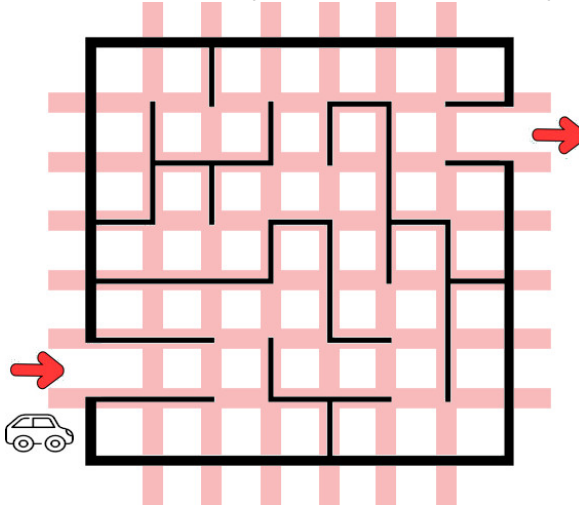
# Apresentação e boas-vindas

# Apresentação e boas-vindas

## Olá?

# Desafio

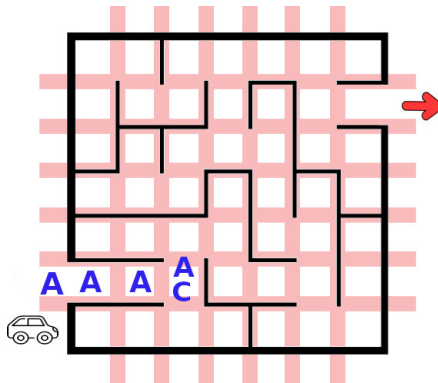
\* A = Passo a frente, \* B = Vire para a direita, \* C = Vire para a esquerda





# 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

# Lógica 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:

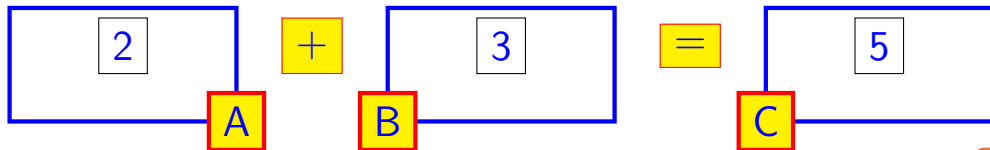
- Passos executados até se atingir o objetivo ou solução de um problema.
- Podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

# Lógica 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**.

# Lógica 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.



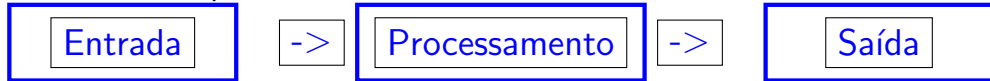
# Lógica de Programação - Programa

- **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.

# Lógica 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.



# Lógica 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



# Lógica 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

```
1 def somar(primeiro_numero: int, segundo_numero: int) -> int:
2     return primeiro_numero + segundo_numero
3
4 # testes da classe
5 #if __name__ == "__main__":
6     # print('total da soma: {}'.format(somar(11,20)))
```

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

```
1 from src import somar
2 import pytest
3
4 @pytest.mark.parametrize(('primeiro_numero', 'segundo_numero', 'resultado_esperado'), [(1, 1, 2), (11, 1, 12)])
5 def teste_somar_com_sucesso(primeiro_numero, segundo_numero, resultado_esperado):
6     assert somar(primeiro_numero, segundo_numero) == resultado_esperado
7
8 @pytest.mark.parametrize(('primeiro_numero', 'segundo_numero', 'resultado_esperado'), [(1, 1, 3), (11, 1, 11)])
9 def teste_soma_com_erro(primeiro_numero, segundo_numero, resultado_esperado):
10     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 prefixo 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 =====
```

→ ↺ 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%
<b>Total</b>	<b>122</b>	<b>0</b>	<b>0</b>	<b>100%</b>

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()

```
1 def obter_numero() -> int:
2     return (int(input("Digite um numero inteiro: ")))
3
4 def obter_tres_numeros_simples() -> list:
5     return [obter_numero(), obter_numero(), obter_numero()]
6
7 def obter_trinta_numeros() -> list:
8     resultado: list = []
9     while len(resultado) < 30:
10         resultado.append(obter_numero())
11     return resultado
12
13 def obter_n_numeros(quantidade: int = 90) -> list:
14     resultado: list = []
15     while len(resultado) < quantidade:
16         resultado.append(obter_numero())
17     return resultado
```

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

# Pytest

```
1 from src import obtem_numero, obter_tres_numeros_simples, obter_trinta_numeros, obter_n_numeros
2 import pytest
3 from unittest.mock import patch
4
5 @pytest.mark.parametrize("valor_esperado", [(10), (9)])
6 def teste_obter_numero_com_sucesso(valor_esperado):
7     with patch('builtins.input', return_value=valor_esperado):
8         resultado = obtem_numero()
9         assert resultado == valor_esperado
10
11 @pytest.mark.parametrize("valor_esperado", [[[10,10,10]], [[9,9,9]]])
12 def teste_obter_tres_numeros_simples_com_sucesso(valor_esperado):
13     with patch('builtins.input', side_effect=valor_esperado):
14         resultado = obter_tres_numeros_simples()
15         assert resultado == valor_esperado
16
17 @pytest.mark.parametrize("valor_esperado", [[[list(range(0, 30))]])])
18 def teste_obter_trinta_numeros_com_sucesso(valor_esperado):
19     with patch('builtins.input', side_effect=valor_esperado):
20         resultado = obter_trinta_numeros()
21         assert resultado == valor_esperado
22
23 @pytest.mark.parametrize("valor_esperado", [[list(range(0, 90))]])
24 def teste_obter_n_numeros_com_sucesso(valor_esperado):
25     with patch('builtins.input', side_effect=valor_esperado):
26         resultado = obter_n_numeros(len(valor_esperado))
27         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()

```
1 from src.codigo_008_input import obter_numero, obter_tres_numeros_simples, obter_trinta_numeros, obter_n_numeros
2
3 def print_obter_numero():
4     print(obter_numero())
5
6 def print_obter_tres_numeros_simples():
7     print(obter_tres_numeros_simples())
8
9 def print_obter_trinta_numeros():
10    print(obter_trinta_numeros())
11
12 def print_obter_n_numeros(n: int = 90):
13    print(obter_n_numeros(n))
```

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

# Pytest

```

1 from src import print_obtem_numero, print_obter_tres_numeros_simples, print_obter_trinta_numeros, print_obter_n_numeros
2 import pytest
3 from unittest.mock import patch
4
5 @pytest.mark.parametrize("valor_esperado", [(10), (9)])
6 def teste_print_obtem_numero_com_sucesso(valor_esperado):
7     with patch('builtins.input', return_value=valor_esperado), patch('builtins.print') as mock_print:
8         print_obtem_numero()
9         mock_print.assert_called_once_with(valor_esperado)
10        assert mock_print.call_count == 1
11
12 @pytest.mark.parametrize("valor_esperado", [[[10,10,10]], ([9,9,9])])
13 def teste_print_obtem_tres_numeros_com_sucesso(valor_esperado):
14     with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
15         print_obter_tres_numeros_simples()
16         mock_print.assert_called_once_with(valor_esperado)
17         assert mock_print.call_count == 1
18
19 @pytest.mark.parametrize("valor_esperado", [([list(range(0, 30))]))],
20                          (list(range(0, 30))))]
21 def teste_print_obtem_trinta_numeros_com_sucesso(valor_esperado):
22     with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
23         print_obter_trinta_numeros()
24         mock_print.assert_called_once_with(valor_esperado)
25         assert mock_print.call_count == 1
26
27 @pytest.mark.parametrize("valor_esperado", [(list(range(0, 90)))]])
28 def teste_print_obtem_n_numeros_com_sucesso(valor_esperado):
29     with patch('builtins.input', side_effect=valor_esperado), patch('builtins.print') as mock_print:
30         print_obter_n_numeros(len(valor_esperado))
31         mock_print.assert_called_once_with(valor_esperado)
32         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)

```
1 somar = lambda primeiro_numero , segundo_numero: primeiro_numero + segundo_numero
2
3 # print('total da soma: '.format(soma(11,20)))
4
5 # ou
6
7 # 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

```

1 from src import somar as somar_lambda
2
3 def teste_somar_com_sucesso():
4     assert somar_lambda(1,2) == 3
5
6 def teste_soma_com_erro():
7     assert somar_lambda(1,2) != 2

```

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

```

$ 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 4 items

test_codigo_001_somar.py .. [ 50%]
test_codigo_002_lambda_somar.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%
test_codigo_001_somar.py             5      0   100%
test_codigo_002_lambda_somar.py      5      0   100%
TOTAL                               13      0   100%

4 passed in 0.03s

```

# 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 ( $\text{saldo atual} = \text{saldo} - \text{débito} + \text{crédito}$ ). 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.
- especificações 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 Responsibility 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 Responsibility 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  $o_1$  do tipo  $S$  há um objeto  $o_2$  do tipo  $T$  de forma que, para todos os programas  $P$  definidos em termos de  $T$ , o comportamento de  $P$  é inalterado quando  $o_1$  é substituído por  $o_2$  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).

# P00

# 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 também 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

# ATENÇÃO

Vamos escrever um algoritmo

# 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

**ATENÇÃO**

**Respirem fundo  
que vamos...**

# Exemplos de criação de classe

```
1 class Produto:
2     serie: int # público
3     __lote: int # privado
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
5         self.nome = nome
6         self.codigo = codigo
7         self.preco = preco
8         self.quantidade = quantidade
9         self.serie = round(codigo * 1000)
10        self.__lote = round(1000)
11    def obtem_nome(self) -> str:
12        return self.nome
13    def obtem_codigo(self) -> int:
14        return self.codigo
15    def obtem_preco(self) -> int:
16        return self.preco
17    def obtem_serie(self) -> int:
18        return self.serie
19    def obtem_lote(self) -> int:
20        return self.__lote
21    def altera_preco(self, novo_preco) -> bool:
22        preco_atual = self.preco
23        self.preco = novo_preco
24        if novo_preco > preco_atual: return True
25        return False
26    def altera_quantidade(self, novo_pedido: int) -> bool:
27        if novo_pedido > self.quantidade: return False
28        self.quantidade -= novo_pedido
29        return True
30    def obtem_produto(self) -> str:
31        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.

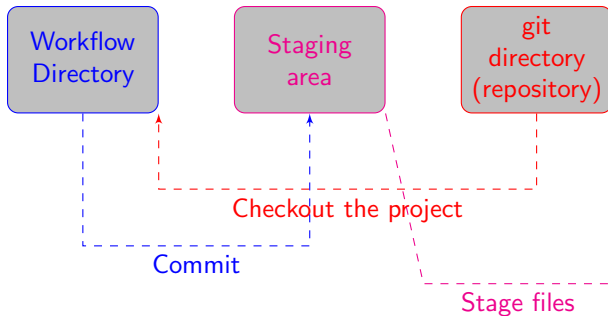
# Como o Git funciona?

- 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

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

*Required fields are marked with an asterisk (\*).*

**Owner \*** **Repository name \***

brunom4ciel / meu-repositorio

meu-repositorio is available.

Great repository names are short and memorable. Need inspiration? How about [legendary-memory](#)?

**Description (optional)**

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs](#).

**Add .gitignore**  
.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

**Choose a license**  
License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

☐ You are creating a public repository in your personal account.

**Create repository**

# 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.

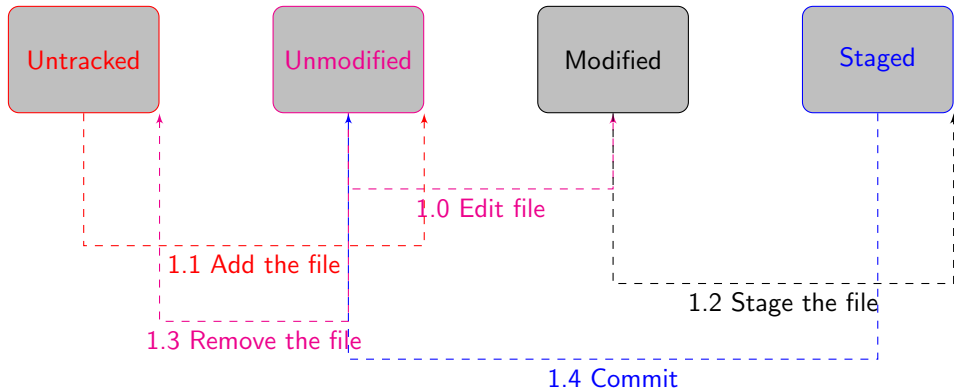
## Adicionando arquivos e o primeiro commit

```
$ 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 branches?

■ `$ 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*

# P00

# Exemplos de criação de classe

## ...continuando.

# Atenção

**ATENÇÃO**

**Respirem fundo  
que vamos...**

# Exemplos de criação de classe

```
1 class Produto:
2     serie: int # público
3     __lote: int # privado
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
5         self.nome = nome
6         self.codigo = codigo
7         self.preco = preco
8         self.quantidade = quantidade
9         self.serie = round(codigo * 1000)
10        self.__lote = round(1000)
11    def obtem_nome(self) -> str:
12        return self.nome
13    def obtem_codigo(self) -> int:
14        return self.codigo
15    def obtem_preco(self) -> int:
16        return self.preco
17    def obtem_serie(self) -> int:
18        return self.serie
19    def obtem_lote(self) -> int:
20        return self.__lote
21    def altera_preco(self, novo_preco) -> bool:
22        preco_atual = self.preco
23        self.preco = novo_preco
24        if novo_preco > preco_atual: return True
25        return False
26    def altera_quantidade(self, novo_pedido: int) -> bool:
27        if novo_pedido > self.quantidade: return False
28        self.quantidade -= novo_pedido
29        return True
30    def obtem_produto(self) -> str:
31        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 possível "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")
```

Py

```
1 from src import Produto
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
5 def testa_criacao_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
6     produto = Produto(nome, codigo, preco, quantidade)
7     assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
8         produto.obtem_lote() != 0 and produto.obtemSerie() != 0
9
10 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
11 def teste_alteracao_preco_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
12     produto = Produto(nome, codigo, preco, quantidade)
13     assert produto.altera_preco(2) == True
14
15 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
16 def teste_alteracao_preco_de_produtos_com_falha(nome, codigo, preco, quantidade):
17     produto = Produto(nome, codigo, preco, quantidade)
18     assert produto.altera_preco(1) == False
19
20 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
21 def teste_alteracao_quantidade_de_produtos_com_sucesso(nome, codigo, preco, quantidade):
22     produto = Produto(nome, codigo, preco, quantidade)
23     assert produto.altera_quantidade(2) == False
24
25 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
26 def teste_alteracao_quantidade_de_produtos_com_falha(nome, codigo, preco, quantidade):
27     produto = Produto(nome, codigo, preco, quantidade)
28     assert produto.altera_quantidade(1) == True
29
30 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
31 def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade):
32     produto = Produto(nome, codigo, preco, quantidade)
33     assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtemSerie(), nome, codigo,
34         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 acoplamento.

\*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 evitar 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

```
1 class Singleton(object):
2     def __new__(cls, *args, **kwargs):
3         if not hasattr(cls, 'instance'):
4             cls.instance = super(Singleton, cls).__new__(cls)
5             cls.instance._initialized = False
6         return cls.instance
7
8     def __init__(self, wrapped_class=None, *args, **kwargs):
9         if wrapped_class and not self._initialized:
10             self._initialized = True
11             self.wrapped_class_instance = wrapped_class(*args, **kwargs)
12
13     def update_instance(self, *args, **kwargs):
14         if hasattr(self, 'wrapped_class_instance'):
15             for key, value in kwargs.items():
16                 setattr(self.wrapped_class_instance, key, value)
17
18     def __getattr__(self, name):
19         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

```

1 import pytest
2 from src import Singleton
3 from src import Produto
4
5 def testa_classe_singleton_se_duas_instancias_sao_iguais_com_sucesso():
6     singleton = Singleton()
7     assert id(singleton) == id(Singleton())
8     # id() Retorna a identidade de um objeto.
9     # Isso é garantido como único entre objetos existentes simultaneamente.
10    # (CPython usa o endereço de memória do objeto.)
11
12    @pytest.mark.parametrize(('produto', 'nome', 'codigo', 'preco', 'quantidade',
13                              'produto2', 'nome2', 'codigo2', 'preco2', 'quantidade2'),
14                              [(Produto, 'Arroz', 1, 1, 1, Produto, 'Feijao', 1, 1, 1),
15                               (Produto, 'Feijao', 1, 1, 1, Produto, 'Arroz', 1, 1, 1)])
16    def testa_classe_singleton_se_duas_instancias_de_produto_sao_iguais_com_sucesso(
17        produto, nome, preco, quantidade, produto2, nome2, codigo2, preco2, quantidade2):
18        produto_inst = Singleton(produto, nome, codigo, preco, quantidade)
19        produto_inst2 = Singleton(produto2, nome2, codigo2, preco2, quantidade2)
20        assert produto_inst.obtem_nome() == produto_inst2.obtem_nome() \
21            and produto_inst.obtem_nome() == 'Arroz' and produto_inst2.obtem_nome() == 'Arroz'
22
23    @pytest.mark.parametrize(('produto', 'nome', 'codigo', 'preco', 'quantidade',
24                              'produto2', 'nome2', 'codigo2', 'preco2', 'quantidade2'),
25                              [(Produto, 'Arroz', 1, 1, 1, Produto, 'Feijao', 1, 1, 1),
26                               (Produto, 'Feijao', 1, 1, 1, Produto, 'Arroz', 1, 1, 1)])
27    def testa_classe_singleton_atualizando_dados_na_instancia_com_sucesso(
28        produto, nome, codigo, preco, quantidade, produto2, nome2, codigo2, preco2, quantidade2):
29        produto_inst = Singleton(produto, nome, codigo, preco, quantidade)
30        produto_inst2 = Singleton(produto2, nome2, codigo2, preco2, quantidade2)
31        produto_inst.update_instance(produto=produto2, nome=nome2, codigo=codigo2,
32                                    preco=preco2, quantidade=quantidade2)
33        assert produto_inst.obtem_nome() == nome2 and produto_inst2.obtem_nome() == nome2

```

# 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 com recursos reduzidos e criá-los somente quando houver necessidade.

# Padrão de projeto Singleton Lazy

```
1 class SingletonLazy:
2     __instance = None
3     @classmethod
4     def get_instance(cls):
5         if not cls.__instance:
6             cls.__instance = SingletonLazy()
7         return cls.__instance
8
9     # Um classmethod é um método que recebe a classe como primeiro argumento
10    # em vez da instância. Isso significa que você pode acessar atributos e
11    # métodos de nível de classe a partir deste método. Um uso comum de
12    # classmethod é para definir construtores alternativos ou implementar
13    # método que gerencia instancias de classe.
14
15    # Neste exemplo, get_instance é um método de classe que nos permite pegar
16    # uma instância da classe por meio do atributo __instance.
17    # 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
18    # 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

```
1 from src import SingletonLazy
2
3 def testa_classe_singleton_lazy_se_duas_instancias_sao_iguais_com_sucesso():
4     singleton = SingletonLazy().get_instance()
5     assert id(singleton) == id(SingletonLazy().get_instance())
6     # id() Retorna a identidade de um objeto.
7     # Isso é garantido como único entre objetos existentes simultaneamente.
8     # (CPython usa o endereço de memória do objeto.)
```

## Código 15: Cobertura de testes da classe Singleton



# P00

## 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

```
1 import sys
2
3 def iniciar_app(argv) -> None:
4     for value in argv:
5         print(value)
6
7 if __name__ == '__main__':
8     iniciar_app(sys.argv)
```

Código 16: Inicialização de script com parâmetros

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

# Pytest

```
1 from src import iniciar_app
2 import pytest
3 from unittest.mock import patch
4 from io import StringIO
5 import sys
6
7 def test_parse_args():
8     original_stdout = sys.stdout
9     sys.stdout = StringIO() # Redireciona stdout para capturar a saída
10    test_argv = ['arg1', 'arg2', 'arg3'] # Argumentos de teste
11    iniciar_app(test_argv)
12    output = sys.stdout.getvalue() # Obtém a saída capturada
13    sys.stdout = original_stdout # Restaura o stdout original
14    expected_output = 'arg1\narg2\narg3\n' # Verifica se a saída está correta
15    assert output == expected_output
16
17 @pytest.mark.parametrize(
18     "test_argv, expected_output",
19     [(['arg1', 'arg2', 'arg3'], 'arg1\narg2\narg3\n'), (['foo', 'bar'], 'foo\nbar\n')]
20 )
21 def test_parse_args_with_patch(test_argv, expected_output):
22     with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
23         iniciar_app(test_argv) # Chama a função com os argumentos de teste
24         output = mock_stdout.getvalue() # Obtém a saída capturada
25         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 ( $\text{saldo atual} = \text{saldo} - \text{débito} + \text{crédito}$ ). 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

```
1 import abc
2
3 class CadeiraAbstrata(metaclass=abc.ABCMeta):
4     @abc.abstractmethod
5     def sentar_em(self):
6         pass
```

Código 18: Código da Classe Cadeira Abstrata

```
1 import abc
2
3 class SofaAbstrato(metaclass=abc.ABCMeta):
4     @abc.abstractmethod
5     def sentar_em(self):
6         pass
```

Código 19: Código da Classe Sofa Abstrato

# Padrão de projeto Abstract Factory

```
1 from src import CadeiraAbstrata
2
3 class CadeiraClassicaConcreta(CadeiraAbstrata):
4     def sentar_em(self):
5         return "Sentado em uma cadeira classica"
```

Código 20: Código da Classe Cadeira Concreta

```
1 from src import SofaAbstrato
2
3 class SofaClassicoConcreta(SofaAbstrato):
4     def sentar_em(self):
5         return "Sentado em um sofa classico"
```

Código 21: Código da Classe Sofa Concreta

# Padrão de projeto Abstract Factory

```
1 from src import CadeiraAbstrata
2
3 class CadeiraModernaConcreta(CadeiraAbstrata):
4     def sentar_em(self):
5         return "Sentado em uma cadeira moderna"
```

Código 22: Código da Classe Cadeira Moderna

```
1 from src import SofaAbstrato
2
3 class SofaModernoConcreta(SofaAbstrato):
4     def sentar_em(self):
5         return "Sentado em um sofa moderno"
```

Código 23: Código da Classe Sofa Moderna

# Padrão de projeto Abstract Factory

```
1 import abc
2 from src import CadeiraAbstrata, SofaAbstrato
3
4 class MoveisFactoryAbstrato(metaclass=abc.ABCMeta):
5     @abc.abstractmethod
6     def criar_cadeira(self) -> CadeiraAbstrata:
7         pass
8
9     @abc.abstractmethod
10    def criar_sofa(self) -> SofaAbstrato:
11        pass
```

Código 24: Código da Classe Moveis Factory Abstrato

# Padrão de projeto Abstract Factory

```
1 from src import CadeiraAbstrata, SofaAbstrato, MoveisFactoryAbstrato, SofaClassicoConcreta, CadeiraClassicaConcreta
2
3 class MoveisClassicosFactoryConcreto(MoveisFactoryAbstrato):
4     def criar_cadeira(self) -> CadeiraAbstrata:
5         return CadeiraClassicaConcreta()
6
7     def criar_sofa(self) -> SofaAbstrato:
8         return SofaClassicoConcreta()
```

Código 25: Código da Classe Cadeira Classica Factory Concretra

```
1 from src import CadeiraAbstrata, SofaAbstrato, MoveisFactoryAbstrato, SofaModernoConcreta, CadeiraModernaConcreta
2
3 class MoveisModernosFactoryConcreto(MoveisFactoryAbstrato):
4     def criar_cadeira(self) -> CadeiraAbstrata:
5         return CadeiraModernaConcreta()
6
7     def criar_sofa(self) -> SofaAbstrato:
8         return SofaModernoConcreta()
```

Código 26: Código da Classe Sofa Classica Factory Concretra

# Padrão de projeto Abstract Factory

```
1 from src import MoveisFactoryAbstrato
2
3 # Cliente usando a Abstract Factory
4 def codigo_cliente(factory: MoveisFactoryAbstrato):
5     cadeira = factory.criar_cadeira()
6     sofa = factory.criar_sofa()
7
8     print(cadeira.sentar_em())
9     print(sofa.sentar_em())
```

Código 27: Código da função código do cliente

# Pytest

```
1 import pytest
2 from unittest.mock import patch
3 from src import MoveisModernosFactoryConcreto, MoveisClassicosFactoryConcreto, codigo_cliente
4
5 @pytest.mark.parametrize("valor_esperado", [[ 'Sentado em uma cadeira moderna', 'Sentado em um sofa moderno' ]])
6 def teste_codigo_cliente_para_usar_moveis_modernos_com_sucesso(valor_esperado):
7     with patch('builtins.print') as mock_print:
8         moderna_factory = MoveisModernosFactoryConcreto()
9         codigo_cliente(moderna_factory)
10        mock_print.assert_any_call(valor_esperado[0])
11        mock_print.assert_any_call(valor_esperado[1])
12        assert mock_print.call_count == len(valor_esperado)
13
14 @pytest.mark.parametrize("valor_esperado", [[ 'Sentado em uma cadeira classica', 'Sentado em um sofa classico' ]])
15 def teste_codigo_cliente_para_usar_moveis_classicos_com_sucesso(valor_esperado):
16     with patch('builtins.print') as mock_print:
17         moderna_factory = MoveisClassicosFactoryConcreto()
18         codigo_cliente(moderna_factory)
19        mock_print.assert_any_call(valor_esperado[0])
20        mock_print.assert_any_call(valor_esperado[1])
21        assert mock_print.call_count == len(valor_esperado)
```

Código 28: Cobertura de testes da função codigo cliente

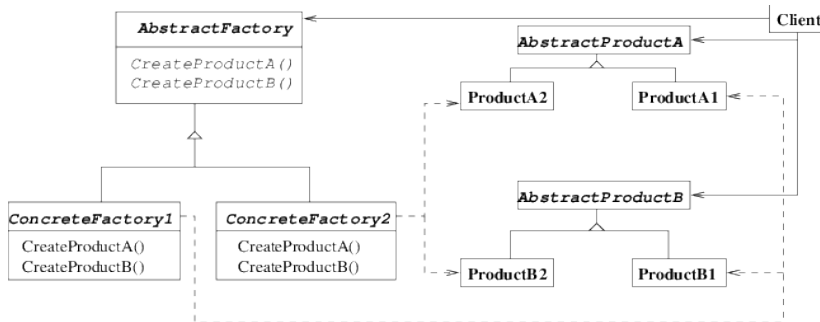


# Padrões de criação - Abstract Factory

## Resumo

1. O padrão Abstract Factory permite a criação de famílias de objetos relacionados (por exemplo, móveis modernos ou clássicos) 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.



# P00

# 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

```
1 from src.codigo_003_classe_produto import Produto
2
3 class ProdutoCritico(Produto):
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, estoque_min: int):
5         super().__init__(nome, codigo, preco, quantidade)
6         self.estoque_min = estoque_min
7
8     def obtem_serie(self) -> int:
9         return self.serie
10
11     def altera_quantidade(self, novo_pedido: int) -> bool:
12         if novo_pedido + self.estoque_min > self.quantidade:
13             return False
14         return super().altera_quantidade(novo_pedido)
15
16     def altera_preco(self, novo_preco: int) -> bool:
17         preco_atual = self.preco
18         if novo_preco > 1.1 * preco_atual or novo_preco < 0.9 * preco_atual:
19             return False
20         super().altera_preco(novo_preco)
21         return True
22
23     def obtem_produto(self) -> str:
24         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

```

1 from src import ProdutoCritico
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
5 def teste_alteracao_preco_de_produtos_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
6     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
7     assert produto_critico.altera_preco(11) == True
8
9 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
10 def teste_alteracao_preco_de_produtos_com_falha(nome, codigo, preco, quantidade, estoque_min):
11     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
12     assert produto_critico.altera_preco(12) == False
13
14 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 10, 20)])
15 def teste_alteracao_quantidade_de_produtos_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
16     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
17     assert produto_critico.altera_quantidade(2) == False
18
19 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
20 def teste_alteracao_quantidade_de_produtos_com_falha(nome, codigo, preco, quantidade, estoque_min):
21     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
22     assert produto_critico.altera_quantidade(1) == True
23
24 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
25 def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
26     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
27     assert produto_critico.obtem_produto() == '{}-{}-{}-{}-{}'.format(nome, codigo, preco, quantidade, estoque_min)
28
29 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'estoque_min'), [('Arroz', 1, 10, 100, 20)])
30 def teste_obtemSerie_com_sucesso(nome, codigo, preco, quantidade, estoque_min):
31     produto_critico = ProdutoCritico(nome, codigo, preco, quantidade, estoque_min)
32     assert produto_critico.obtemSerie() == 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-backend
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'.



## 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

```
1 import abc
2 class ProdutoAbstrato(metaclass=abc.ABCMeta):
3     @abc.abstractmethod
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
5         pass
6     @abc.abstractmethod
7     def obtem_nome(self) -> str:
8         pass
9     @abc.abstractmethod
10    def obtem_codigo(self) -> int:
11        pass
12    @abc.abstractmethod
13    def obtem_preco(self) -> int:
14        pass
15    @abc.abstractmethod
16    def obtem_serie(self) -> int:
17        pass
18    @abc.abstractmethod
19    def obtem_lote(self) -> int:
20        pass
21    @abc.abstractmethod
22    def altera_preco(self, novo_preco) -> bool:
23        pass
24    @abc.abstractmethod
25    def altera_quantidade(self, novo_pedido: int) -> bool:
26        pass
27    @abc.abstractmethod
28    def obtem_produto(self) -> str:
29        pass
```

Código 31: Classe Produto Abstrata

\*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

```
1 from src import ProdutoAbstrato
2
3 class ProdutoConcreto(ProdutoAbstrato):
4     serie: int # público
5     __lote: int # privado
6     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
7         self.nome = nome
8         self.codigo = codigo
9         self.preco = preco
10        self.quantidade = quantidade
11        self.serie = round(codigo * 1000)
12        self.__lote = round(1000)
13    def obtem_nome(self) -> str:
14        return self.nome
15    def obtem_codigo(self) -> int:
16        return self.codigo
17    def obtem_preco(self) -> int:
18        return self.preco
19    def obtem_serie(self) -> int:
20        return self.serie
21    def obtem_lote(self) -> int:
22        return self.__lote
23    def altera_preco(self, novo_preco) -> bool:
24        preco_atual = self.preco
25        self.preco = novo_preco
26        if novo_preco > preco_atual: return True
27        return False
28    def altera_quantidade(self, novo_pedido: int) -> bool:
29        if novo_pedido > self.quantidade: return False
30        self.quantidade -= novo_pedido
31        return True
32    def obtem_produto(self) -> str:
33        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

## Código 32: Classe Produto Abstrato

# Pytest

```
1 from src import ProdutoConcreto
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
5 def testa_classe_produto_concreto_geral_com_sucesso(nome, codigo, preco, quantidade):
6     produto = ProdutoConcreto(nome, codigo, preco, quantidade)
7     assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
8         produto.obtem_lote() != 0 and produto.obtemSerie() != 0
9     assert produto.obtem_produto() == '{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtemSerie(), nome, codigo,
10         preco, quantidade)
11     assert produto.altera_preco(2) == True
12     assert produto.altera_preco(1) == False
13     assert produto.altera_quantidade(2) == False
14     assert produto.altera_quantidade(1) == True
```

Código 33: Cobertura de testes da classe Produto Concreto



# Pytest

```

----- 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

```
1 class ProdutoAbstratoExcecao:
2
3     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
4         raise NotImplementedError()
5     def obtem_nome(self) -> str:
6         raise NotImplementedError()
7     def obtem_codigo(self) -> int:
8         raise NotImplementedError()
9     def obtem_preco(self) -> int:
10        raise NotImplementedError()
11    def obtem_serie(self) -> int:
12        raise NotImplementedError()
13    def obtem_lote(self) -> int:
14        raise NotImplementedError()
15    def altera_preco(self, novo_preco) -> bool:
16        raise NotImplementedError()
17    def altera_quantidade(self, novo_pedido: int) -> bool:
18        raise NotImplementedError()
19    def obtem_produto(self) -> str:
20        raise NotImplementedError()
```

Código 34: Classe Produto Abstrato Excecao

# Interface com Classes Abstratas - Caso com Exceção

```
1 from src import ProdutoAbstratoExcecao
2
3 class ProdutoConcretoExcecao(ProdutoAbstratoExcecao):
4     serie: int # público
5     __lote: int # privado
6     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
7         self.nome = nome
8         self.codigo = codigo
9         self.preco = preco
10        self.quantidade = quantidade
11        self.serie = round(codigo * 1000)
12        self.__lote = round(1000)
13    def obtem_nome(self) -> str:
14        return self.nome
15    def obtem_codigo(self) -> int:
16        return self.codigo
17    def obtem_preco(self) -> int:
18        return self.preco
19    def obtem_serie(self) -> int:
20        return self.serie
21    def obtem_lote(self) -> int:
22        return self.__lote
23    def altera_preco(self, novo_preco) -> bool:
24        preco_atual = self.preco
25        self.preco = novo_preco
26        if novo_preco > preco_atual: return True
27        return False
28    def altera_quantidade(self, novo_pedido: int) -> bool:
29        if novo_pedido > self.quantidade: return False
30        self.quantidade -= novo_pedido
31        return True
32    def obtem_produto(self) -> str:
33        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

# Pytest

```

1 from src import ProdutoConcretoExcecao
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [
5     ('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
6 def testa_classe_produto_concreto_excecao_geral_com_sucesso(nome, codigo, preco, quantidade):
7     produto = ProdutoConcretoExcecao(nome, codigo, preco, quantidade)
8     assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
9         produto.obtem_lote() != 0 and produto.obtemSerie() != 0
10    assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtemSerie(), nome, codigo,
11        preco, quantidade)
12    assert produto.altera_preco(2) == True
13    assert produto.altera_preco(1) == False
14    assert produto.altera_quantidade(2) == False
15    assert produto.altera_quantidade(1) == True

```

## Código 36: Cobertura de testes da classe Produto Concreto Exceção

# 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.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

```
1 class Veiculo:
2     chassi: int # público
3     __motor: int # privado
4     def __init__(self, chassi: int, motor: int):
5         self.chassi = chassi
6         self.__motor = motor
7     def obtem_chassi(self) -> int:
8         return self.chassi
9     def obtem_motor(self) -> int:
10        return self.__motor
11    def altera_chassi(self, novo_chassi: int) -> bool:
12        try:
13            if novo_chassi != self.chassi:
14                self.chassi = novo_chassi
15                return True
16            else:
17                raise ValueError("Chassi igual ao atual")
18        except ValueError as error:
19            raise ValueError(error)
```

Código 37: Tratamento de exceções

# Pytest

```
1 from src import Veiculo
2 import pytest
3
4 @pytest.mark.parametrize(('chassi', 'motor'), [('KHG969878976G7DF9G7', 'SGD97S9')])
5 def teste_obtem_chassi_e_motor_com_sucesso(chassi, motor):
6     veiculo = Veiculo(chassi, motor)
7     assert veiculo.obtem_chassi() == chassi and veiculo.obtem_motor() == motor
8
9 @pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G7')])
10 def teste_alterar_chassis_iguais(chassi, motor, novo_chassi):
11     veiculo = Veiculo(chassi, motor)
12     with pytest.raises(ValueError) as excinfo:
13         veiculo.altera_chassi(novo_chassi)
14     assert str(excinfo.value) == "Chassi igual ao atual" and excinfo.type is ValueError
15
16 @pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G6')])
17 def teste_alterar_chassis_diferentes(chassi, motor, novo_chassi):
18     veiculo = Veiculo(chassi, motor)
19     assert veiculo.altera_chassi(novo_chassi) == True
```

## Código 38: 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

```
1 from src.codigo_003_classe_produto import Produto
2
3 class ProdutoPercivel(Produto):
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, validade: int):
5         super().__init__(nome, codigo, preco, quantidade)
6         self.validade = validade
7
8     def obter_produto(self) -> str:
9         return '{}-{}-{}-{}-{}'.format(self.nome, self.codigo, self.preco, self.quantidade, self.validade)
```

Código 39: Polimorfismo de classe

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

# Pytest

```
1 from src import ProdutoPerecivel
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'validade'), [('Arroz', 1, 10, 100, 20)])
5 def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, validade):
6     produto_perecivel = ProdutoPerecivel(nome, codigo, preco, quantidade, validade)
7     assert produto_perecivel.obtem_produto() == '{0}-{0}-{0}-{0}-{0}'.format(nome, codigo, preco, quantidade, validade)
```

## Código 40: 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_repecivel.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 ..
test_codigo_002_lambda_somar.py ..
test_codigo_003_classe_produto.py .....
test_codigo_004_classe_produto_critico.py .....
test_codigo_005_classe_produto_repecivel.py .

----- 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_repecivel.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 anterior

```
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.
```



# 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

```
1 import abc
2
3 class DocumentoAbstrata(metaclass=abc.ABCMeta):
4     @abc.abstractmethod
5     def abrir(self):
6         pass
```

Código 41: Código da Classe Documento - codigo\_016\_documento\_abstrata.py Abstrata

```
1 from src import DocumentoAbstrata
2
3 class DocumentoPdfConcreta(DocumentoAbstrata):
4     def abrir(self):
5         return "Abrindo um documento PDF"
```

Código 42: Código da Classe Documento Pdf - codigo\_016\_documento\_pdf\_concreta.py Concreta

```
1 from src import DocumentoAbstrata
2
3 class DocumentoMsWordConcreta(DocumentoAbstrata):
4     def abrir(self):
5         return "Abrindo um documento Microsoft Word"
```

Código 43: Código da Classe Documento Ms Word - codigo\_016\_documento\_msword\_concreta.py Concreta

# Padrão de projeto Factory Method

```
1 import sys
2 import os
3 # Adiciona o caminho do diretório 'atual' ao sys.path.
4 # Isso resolver problemas com imports de módulos que estão no mesmo diretório.
5 sys.path.append(os.path.join(os.path.dirname(__file__), '..', '..'))
6 from src import DocumentoPdfConcreta
7 from src import DocumentoMsWordConcreta
8
9 # Cliente usando a Factory Method
10 def codigo_cliente_factory_method(tipo_documento: str):
11
12     if(tipo_documento == "word" or tipo_documento == "pdf"):
13
14         if tipo_documento == "word":
15             documento = DocumentoMsWordConcreta()
16
17         elif tipo_documento == "pdf":
18             documento = DocumentoPdfConcreta()
19
20     print(documento.abrir())
21 else:
22
23     print("Tipo de documento desconhecido")
```

Código 44: Código da função codigo\_cliente\_factory\_method

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

# Padrão de projeto Factory Method

```
1 import pytest
2 from unittest.mock import patch
3 from src import codigo_cliente_factory_method
4
5 @pytest.mark.parametrize("valor_esperado", [[("Abrindo um documento PDF")]])
6 def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
7     with patch('builtins.print') as mock_print:
8         codigo_cliente_factory_method("pdf")
9         mock_print.assert_any_call(valor_esperado[0])
10        assert mock_print.call_count == len(valor_esperado)
11
12 @pytest.mark.parametrize("valor_esperado", [[("Abrindo um documento Microsoft Word")]])
13 def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
14     with patch('builtins.print') as mock_print:
15         codigo_cliente_factory_method("word")
16         mock_print.assert_any_call(valor_esperado[0])
17        assert mock_print.call_count == len(valor_esperado)
18
19 @pytest.mark.parametrize("valor_esperado", [[("Tipo de documento desconhecido")]])
20 def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
21     with patch('builtins.print') as mock_print:
22         codigo_cliente_factory_method("outro")
23         mock_print.assert_any_call(valor_esperado[0])
24        assert mock_print.call_count == len(valor_esperado)
```

Código 45: Testes do Código do cliente para utilizar a Factory Method - test\_codigo\_016\_cliente\_factory\_method.py

# Padrão de projeto Factory Method

```
1 from src import DocumentoMsWordConcreta, DocumentoPdfConcreta
2
3 class DocumentoFactoryMethodConcreta:
4
5     def __init__(self, tipo_documento: str):
6         if tipo_documento == "word":
7             self.document = DocumentoMsWordConcreta()
8         elif tipo_documento == "pdf":
9             self.document = DocumentoPdfConcreta()
10
11     def abrir(self):
12         return self.document.abrir()
```

Código 46: 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

```
1 from src import DocumentoFactoryMethodConcreta
2
3 # Cliente usando a Factory Method
4 def codigo_cliente_factory_method2(tipo_documento: str):
5
6     if(tipo_documento == "word" or tipo_documento == "pdf"):
7
8         documento = DocumentoFactoryMethodConcreta(tipo_documento)
9
10        print(documento.abrir())
11
12    else:
13
14        print("Tipo de documento desconhecido")
```

Código 47: Código do cliente para utilizar a Factory Method -  
codigo\_016\_codigo\_cliente\_factory\_method2.py

# Padrão de projeto Factory Method

```
1 import pytest
2 from unittest.mock import patch
3 from src import codigo_cliente_factory_method2
4
5 @pytest.mark.parametrize("valor_esperado", [[("Abrindo um documento PDF")]])
6 def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
7     with patch('builtins.print') as mock_print:
8         codigo_cliente_factory_method2("pdf")
9         mock_print.assert_any_call(valor_esperado[0])
10        assert mock_print.call_count == len(valor_esperado)
11
12 @pytest.mark.parametrize("valor_esperado", [[("Abrindo um documento Microsoft Word")]])
13 def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
14     with patch('builtins.print') as mock_print:
15         codigo_cliente_factory_method2("word")
16         mock_print.assert_any_call(valor_esperado[0])
17        assert mock_print.call_count == len(valor_esperado)
18
19 @pytest.mark.parametrize("valor_esperado", [[("Tipo de documento desconhecido")]])
20 def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
21     with patch('builtins.print') as mock_print:
22         codigo_cliente_factory_method2("outro")
23         mock_print.assert_any_call(valor_esperado[0])
24        assert mock_print.call_count == len(valor_esperado)
```

Código 48: 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.

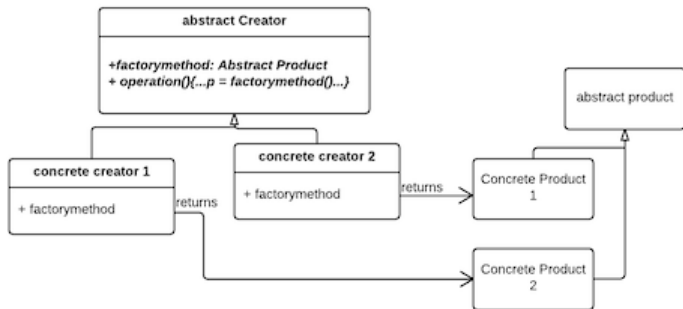
### ■ Complexidade

- 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



# P00

# Exemplos de criação de classe

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

## 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

```
1 import abc
2 class ProdutoAbstrato(metaclass=abc.ABCMeta):
3     @abc.abstractmethod
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
5         pass
6     @abc.abstractmethod
7     def obtem_nome(self) -> str:
8         pass
9     @abc.abstractmethod
10    def obtem_codigo(self) -> int:
11        pass
12    @abc.abstractmethod
13    def obtem_preco(self) -> int:
14        pass
15    @abc.abstractmethod
16    def obtem_serie(self) -> int:
17        pass
18    @abc.abstractmethod
19    def obtem_lote(self) -> int:
20        pass
21    @abc.abstractmethod
22    def altera_preco(self, novo_preco) -> bool:
23        pass
24    @abc.abstractmethod
25    def altera_quantidade(self, novo_pedido: int) -> bool:
26        pass
27    @abc.abstractmethod
28    def obtem_produto(self) -> str:
29        pass
```

Código 49: Classe Produto Abstrata

\*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

```
1 from src import ProdutoAbstrato
2
3 class ProdutoConcreto(ProdutoAbstrato):
4     serie: int # público
5     __lote: int # privado
6     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
7         self.nome = nome
8         self.codigo = codigo
9         self.preco = preco
10        self.quantidade = quantidade
11        self.serie = round(codigo * 1000)
12        self.__lote = round(1000)
13    def obtem_nome(self) -> str:
14        return self.nome
15    def obtem_codigo(self) -> int:
16        return self.codigo
17    def obtem_preco(self) -> int:
18        return self.preco
19    def obtem_serie(self) -> int:
20        return self.serie
21    def obtem_lote(self) -> int:
22        return self.__lote
23    def altera_preco(self, novo_preco) -> bool:
24        preco_atual = self.preco
25        self.preco = novo_preco
26        if novo_preco > preco_atual: return True
27        return False
28    def altera_quantidade(self, novo_pedido: int) -> bool:
29        if novo_pedido > self.quantidade: return False
30        self.quantidade -= novo_pedido
31        return True
32    def obtem_produto(self) -> str:
33        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

## Código 50: Classe Produto Abstrato

# Pytest

```
1 from src import ProdutoConcreto
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
5 def testa_classe_produto_concreto_geral_com_sucesso(nome, codigo, preco, quantidade):
6     produto = ProdutoConcreto(nome, codigo, preco, quantidade)
7     assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
8         produto.obtem_lote() != 0 and produto.obtem_serie() != 0
9     assert produto.obtem_produto() == '{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo,
10         preco, quantidade)
11     assert produto.altera_preco(2) == True
12     assert produto.altera_preco(1) == False
13     assert produto.altera_quantidade(2) == False
14     assert produto.altera_quantidade(1) == True
```

Código 51: Cobertura de testes da classe Produto Concreto

# Pytest

```

----- 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

```
1 class ProdutoAbstratoExcecao:
2
3     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
4         raise NotImplementedError()
5     def obtem_nome(self) -> str:
6         raise NotImplementedError()
7     def obtem_codigo(self) -> int:
8         raise NotImplementedError()
9     def obtem_preco(self) -> int:
10        raise NotImplementedError()
11    def obtem_serie(self) -> int:
12        raise NotImplementedError()
13    def obtem_lote(self) -> int:
14        raise NotImplementedError()
15    def altera_preco(self, novo_preco) -> bool:
16        raise NotImplementedError()
17    def altera_quantidade(self, novo_pedido: int) -> bool:
18        raise NotImplementedError()
19    def obtem_produto(self) -> str:
20        raise NotImplementedError()
```

Código 52: Classe Produto Abstrato Excecao

# Interface com Classes Abstratas - Caso com Exceção

```
1 from src import ProdutoAbstratoExcecao
2
3 class ProdutoConcretoExcecao(ProdutoAbstratoExcecao):
4     serie: int # público
5     __lote: int # privado
6     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
7         self.nome = nome
8         self.codigo = codigo
9         self.preco = preco
10        self.quantidade = quantidade
11        self.serie = round(codigo * 1000)
12        self.__lote = round(1000)
13    def obtem_nome(self) -> str:
14        return self.nome
15    def obtem_codigo(self) -> int:
16        return self.codigo
17    def obtem_preco(self) -> int:
18        return self.preco
19    def obtem_serie(self) -> int:
20        return self.serie
21    def obtem_lote(self) -> int:
22        return self.__lote
23    def altera_preco(self, novo_preco) -> bool:
24        preco_atual = self.preco
25        self.preco = novo_preco
26        if novo_preco > preco_atual: return True
27        return False
28    def altera_quantidade(self, novo_pedido: int) -> bool:
29        if novo_pedido > self.quantidade: return False
30        self.quantidade -= novo_pedido
31        return True
32    def obtem_produto(self) -> str:
33        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

# Pytest

```

1 from src import ProdutoConcretoExcecao
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [
5     ('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
6 def testa_classe_produto_concreto_excecao_geral_com_sucesso(nome, codigo, preco, quantidade):
7     produto = ProdutoConcretoExcecao(nome, codigo, preco, quantidade)
8     assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
9         produto.obtem_lote() != 0 and produto.obtemSerie() != 0
10    assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtemSerie(), nome, codigo,
11        preco, quantidade)
12    assert produto.altera_preco(2) == True
13    assert produto.altera_preco(1) == False
14    assert produto.altera_quantidade(2) == False
15    assert produto.altera_quantidade(1) == True

```

## Código 54: Cobertura de testes da classe Produto Concreto Exceção



# 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

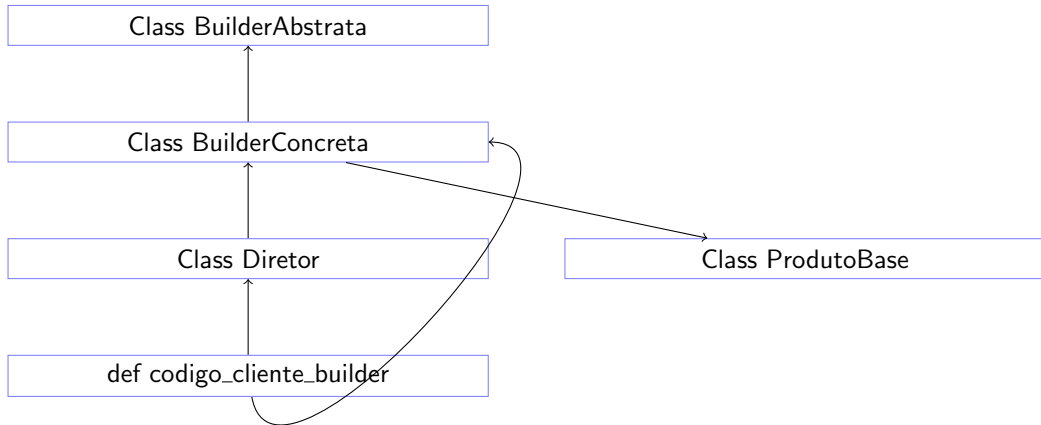
- Produto: O objeto que está sendo construído.
- Builder: Interface que define os métodos de construção.
- BuilderConcreta: Implementa os métodos do Builder e constrói as partes do Produto.
- Diretor: Controla o processo de construção, definindo a ordem em que as partes são construídas.

# Padrões de criação - Builder

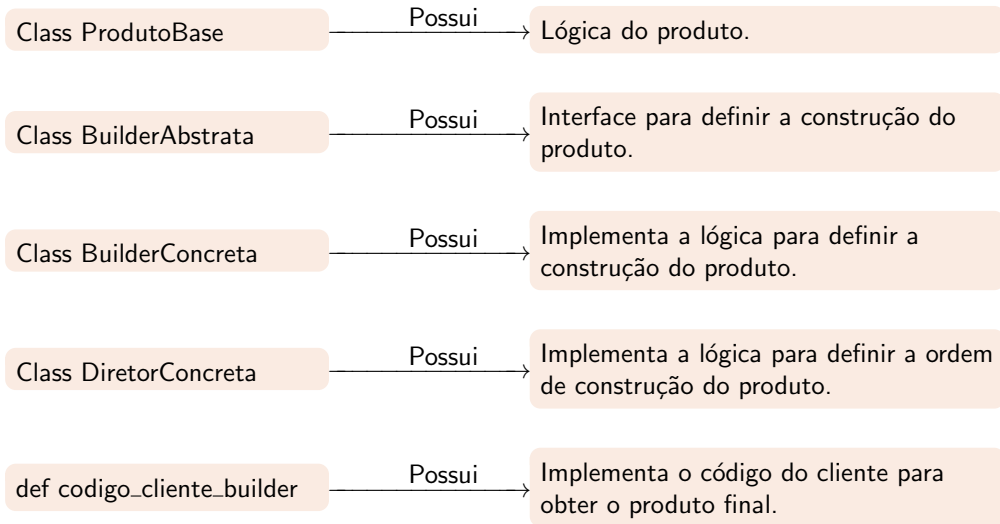
## Aplicação

- 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





# Padrão de projeto - Builder

```
1 from typing import Any
2
3 class ProdutoBase():
4     def __init__(self) -> None:
5         self._partes = []
6
7     def adicionar_parte(self, parte: Any) -> None:
8         self._partes.append(parte)
9
10    def __str__(self):
11        return f"Partes do produto: {' ', ' '.join(self._partes)}"
```

Código 55: Código da Classe Produto Base - codigo\_017\_produto\_base.py

# Padrão de projeto - Builder

```
1 import abc
2 from .codigo_017_produto_base import ProdutoBase
3
4 class BuilderAbstrato(metaclass=abc.ABCMeta):
5
6     @abc.abstractmethod
7     def limpar(self) -> None:
8         pass
9
10    @abc.abstractmethod
11    def produto(self) -> ProdutoBase:
12        pass
13
14    @abc.abstractmethod
15    def construir_parte_a(self) -> None:
16        pass
17
18    @abc.abstractmethod
19    def construir_parte_b(self) -> None:
20        pass
21
22    @abc.abstractmethod
23    def construir_parte_c(self) -> None:
24        pass
```

Código 56: Código da Classe Builder Abtratato - codigo\_017\_builder\_abstrato.py

# Padrão de projeto - Builder

```
1 from .codigo_017_builder_abstrato import BuilderAbstrato
2 from .codigo_017_produto_base import ProdutoBase
3
4 class BuilderConcreta(BuilderAbstrato):
5
6     def __init__(self):
7         self.limpar()
8
9     def limpar(self) -> None:
10         self.__produtoBase = ProdutoBase()
11
12     @property
13     def produto(self) -> ProdutoBase:
14         produto = self.__produtoBase
15         self.limpar()
16         return produto
17
18     def construir_parte_a(self) -> None:
19         self.__produtoBase.adicionar_parte("Parte a")
20
21     def construir_parte_b(self) -> None:
22         self.__produtoBase.adicionar_parte("Parte b")
23
24     def construir_parte_c(self) -> None:
25         self.__produtoBase.adicionar_parte("Parte c")
```

Código 57: Código da Classe Builder Concreta - codigo\_017\_builder\_concreta.py

# Padrão de projeto - Builder

```
1 from .codigo_017_builder_abstrato import BuilderAbstrato
2
3 class Diretor:
4     def __init__(self) -> None:
5         self.__construir = None
6
7     @property
8     def construir(self) -> BuilderAbstrato:
9         return self.__construir
10
11     @construir.setter
12     def construir(self, construir: BuilderAbstrato) -> None:
13         self.__construir = construir
14
15     def construir_produto_simples(self) -> None:
16         self.__construir.construir_parte_c()
17
18     def construir_produto_sofisticado(self) -> None:
19         self.__construir.construir_parte_a()
20         self.__construir.construir_parte_b()
21         self.__construir.construir_parte_c()
```

Código 58: Código da Classe Diretor Concreta - codigo\_017\_diretor\_base.py

# Padrão de projeto - Builder

```
1 # trecho de código para que seja possível você inicializar direto este script
2 # com base nos pacotes.
3 # início
4 import sys
5 import os
6 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..../..'))) # volta 2 pastas
7 # fim
8 from padroesdeprojetos.criacao.codigo_017_builder_concreta import BuilderConcreta
9 from padroesdeprojetos.criacao.codigo_017_diretor_base import Diretor
10
11 def codigo_cliente_builder():
12     diretor = Diretor()
13     diretor.construir = BuilderConcreta()
14
15     diretor.construir_produto_simples()
16     print(f"{diretor.construir.produto}")
17
18     diretor.construir_produto_sofisticado()
19     print(f"{diretor.construir.produto}")
20
21 if __name__ == "__main__":
22     codigo_cliente_builder()
```

Código 59: Código da Função Código do cliente - codigo\_017\_codigo\_cliente\_builder.py

# Pytest

```
1 import pytest
2 from unittest.mock import patch
3 from src.padroesdeprojetos.criacao import codigo_cliente_builder
4
5 @pytest.mark.parametrize("valor_esperado", [
6     ("Partes do produto: Parte c", "Partes do produto: Parte a, Parte b, Parte c")
7 ])
8 def teste_codigo_cliente_builder_com_sucesso(valor_esperado):
9     with patch('builtins.print') as mock_print:
10         codigo_cliente_builder()
11         mock_print.assert_any_call(valor_esperado[0])
12         mock_print.assert_any_call(valor_esperado[1])
13         assert mock_print.call_count == len(valor_esperado)
```

Código 60: 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 sobreescrevem 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.

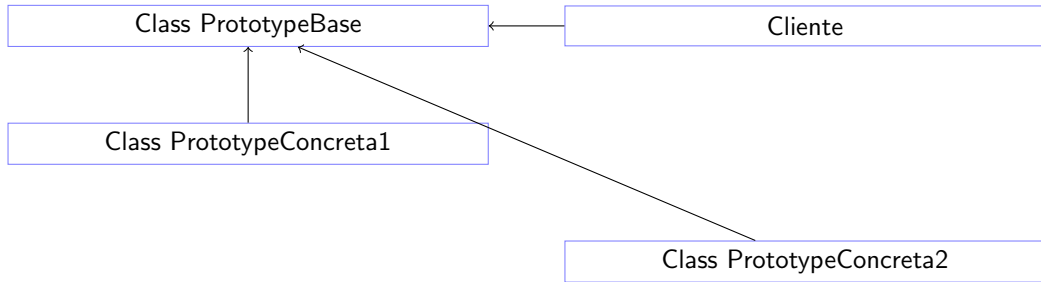
# Padrões de criação - Prototype

## Aplicação

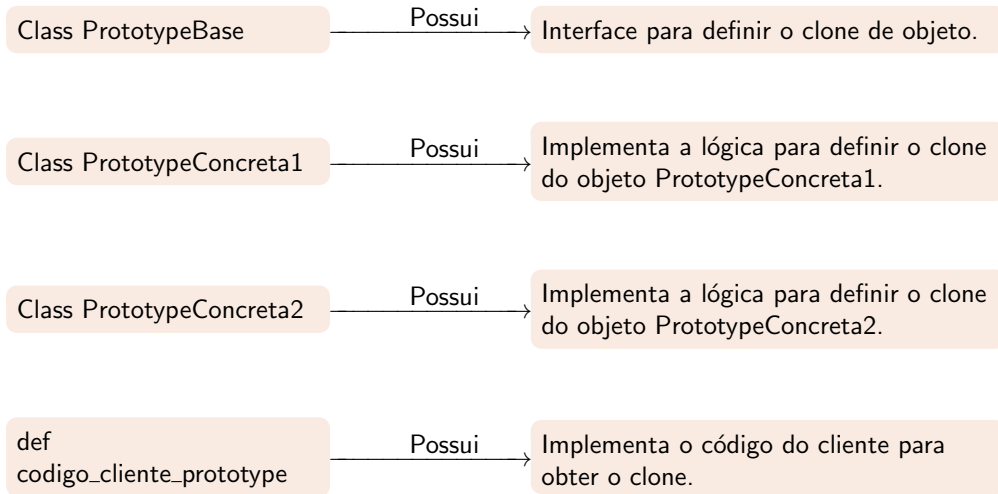
- Quando o custo de criação de um novo objeto é muito caro ou complexo.
- Quando você deseja evitar a duplicação do estado de configuração dos objetos.
- Quando você precisa de uma variedade de objetos semelhantes.

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



# Padrão de projeto - Prototype

```
1 class PrototypeBase():  
2     def clone(self):  
3         """Deve retornar uma copia do objeto."""  
4         pass
```

Código 61: Código da Classe Prototype Base - codigo\_018\_prototype\_base.py

# Padrão de projeto - Prototype

```
1 from .codigo_018_prototype_base import PrototypeBase
2 import copy
3
4 class PrototypeConcreta1(PrototypeBase):
5     def __init__(self, value):
6         self.value = value
7
8     def clone(self):
9         # Usa a função deepcopy para criar uma cópia do objeto
10        # Leia mais em https://docs.python.org/pt-br/3/library/copy.html
11        return copy.deepcopy(self)
12
13    def __str__(self):
14        return f"PrototypeConcreta1 com valor: {self.value}"
```

Código 62: 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

```
1 from .codigo_018_prototype_base import PrototypeBase
2 import copy
3
4 class PrototypeConcreta2(PrototypeBase):
5     def __init__(self, value):
6         self.value = value
7
8     def clone(self):
9         # Usa a função deepcopy para criar uma cópia do objeto
10        # Leia mais em https://docs.python.org/pt-br/3/library/copy.html
11        return copy.deepcopy(self)
12
13    def __str__(self):
14        return f"PrototypeConcreta2 com valor: {self.value}"
```

Código 63: 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

```

1  if __name__ == "__main__":
2      import sys
3      import os
4      sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..'/'..'/'))) # volta 2 pastas
5      from padroesdeprojetos.criacao.codigo_018_prototype_concreta1 import PrototypeConcreta1
6      from padroesdeprojetos.criacao.codigo_018_prototype_concreta2 import PrototypeConcreta2
7  else:
8      from .codigo_018_prototype_concreta1 import PrototypeConcreta1
9      from .codigo_018_prototype_concreta2 import PrototypeConcreta2
10
11 def codigo_cliente_prototype():
12     # Criando os protótipos
13     prototype1 = PrototypeConcreta1("Valor 1")
14     prototype2 = PrototypeConcreta2("Valor 2")
15
16     # Clonando os protótipos
17     clone1 = prototype1.clone()
18     clone2 = prototype2.clone()
19
20     # Exibindo os objetos clonados
21     print(f"{clone1}") # PrototypeConcreta1 com Valor: Valor 1
22     print(f"{clone2}") # PrototypeConcreta2 com Valor: Valor 2
23
24 if __name__ == "__main__":
25     codigo_cliente_prototype()

```

Código 64: Código da Função Código do cliente - codigo\_018\_codigo\_cliente\_prototype.py

# Pytest

```
1 import pytest
2 from unittest.mock import patch
3 from src.padroesdeprojetos.criacao import codigo_cliente_prototype
4
5 @pytest.mark.parametrize("valor_esperado", [{"PrototypeConcreta1 com valor: Valor 1", "PrototypeConcreta2 com valor:
6     Valor 2"}])
7 def teste_codigo_cliente_prototype_com_sucesso(valor_esperado):
8     with patch('builtins.print') as mock_print:
9         codigo_cliente_prototype()
10        mock_print.assert_any_call(valor_esperado[0])
11        mock_print.assert_any_call(valor_esperado[1])
12    assert mock_print.call_count == len(valor_esperado)
```

Código 65: Código do teste para o Código do cliente - test\_codigo\_018\_codigo\_cliente\_prototype.py

# P00

# 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.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

```
1 class Veiculo:
2     chassi: int # público
3     __motor: int # privado
4     def __init__(self, chassi: int, motor: int):
5         self.chassi = chassi
6         self.__motor = motor
7     def obtem_chassi(self) -> int:
8         return self.chassi
9     def obtem_motor(self) -> int:
10        return self.__motor
11    def altera_chassi(self, novo_chassi: int) -> bool:
12        try:
13            if novo_chassi != self.chassi:
14                self.chassi = novo_chassi
15                return True
16            else:
17                raise ValueError("Chassi igual ao atual")
18        except ValueError as error:
19            raise ValueError(error)
```

Código 66: Tratamento de exceções

# Pytest

```
1 from src import Veiculo
2 import pytest
3
4 @pytest.mark.parametrize(('chassi', 'motor'), [('KHG969878976G7DF9G7', 'SGD97S9')])
5 def teste_obtem_chassi_e_motor_com_sucesso(chassi, motor):
6     veiculo = Veiculo(chassi, motor)
7     assert veiculo.obtem_chassi() == chassi and veiculo.obtem_motor() == motor
8
9 @pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G7')])
10 def teste_alterar_chassis_iguais(chassi, motor, novo_chassi):
11     veiculo = Veiculo(chassi, motor)
12     with pytest.raises(ValueError) as excinfo:
13         veiculo.altera_chassi(novo_chassi)
14     assert str(excinfo.value) == "Chassi igual ao atual" and excinfo.type is ValueError
15
16 @pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G6')])
17 def teste_alterar_chassis_diferentes(chassi, motor, novo_chassi):
18     veiculo = Veiculo(chassi, motor)
19     assert veiculo.altera_chassi(novo_chassi) == True
```

## Código 67: 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

```
1 from src.codigo_003_classe_produto import Produto
2
3 class ProdutoPercivel(Produto):
4     def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, validade: int):
5         super().__init__(nome, codigo, preco, quantidade)
6         self.validade = validade
7
8     def obter_produto(self) -> str:
9         return '{}-{}-{}-{}-{}'.format(self.nome, self.codigo, self.preco, self.quantidade, self.validade)
```

Código 68: Polimorfismo de classe

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

# Pytest

```
1 from src import ProdutoPerecivel
2 import pytest
3
4 @pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'validade'), [('Arroz', 1, 10, 100, 20)])
5 def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, validade):
6     produto_perecivel = ProdutoPerecivel(nome, codigo, preco, quantidade, validade)
7     assert produto_perecivel.obtem_produto() == '{0}-{0}-{0}-{0}-{0}'.format(nome, codigo, preco, quantidade, validade)
```

## Código 69: 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_repecivel.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 ..
test_codigo_002_lambda_somar.py ..
test_codigo_003_classe_produto.py .....
test_codigo_004_classe_produto_critico.py .....
test_codigo_005_classe_produto_repecivel.py .

----- 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_repecivel.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 anterior

```
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

**Backend**  
**Python com Django**  
**Prof. Bruno Iran Ferreira Maciel**



**BRUNO**

Aug 2024

---

# Exercícios 1

- Exercício 001** Ler 4 valores (considere que não serão informados valores iguais). Escreva a soma dos dois últimos números. . . . .19
- 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) . . . . .19
- 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. . . . .19
- 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. . . . .19
- 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. . . . .31

## Exercícios 2

- Exercício 006** Ler um valor numérico e escrever o seu antecessor. Ex: Ler  $n = 20$ , Escreva 19. **31**
- 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. . . . . **31**
- 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. . . . . **31**
- Exercício 009** Sistema de ordenação de valores. Ler 5 valores (considere que não serão informados valores iguais). Escrever os números em ordem CRESCENTE. . . . . **32**
- Exercício 010** Sistema de ordenação de valores. Ler 5 valores (considere que não serão informados valores iguais). Escrever os números em ordem DECRESCENTE. . . . . **32**
- 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. . . . . **32**
- 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. . **32**

## Exercícios 3

- 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. . . . .42
- 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. . . . .57
- 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). . . . .57
- 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 ( $\text{saldo atual} = \text{saldo} - \text{débito} + \text{crédito}$ ). Também verificar se saldo atual for maior ou igual a zero, escrever a mensagem 'Saldo Positivo', senão escrever a mensagem 'Saldo Negativo'. . . . . 61, 164

## Exercícios 4

- 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'.* 186
- Exercício 025** *Leia um número inteiro. Escreva o número lido.* . . . . . 49
- Exercício 026** *Leia três números inteiro e guarde em uma lista. Escreva os números da lista.* .50
- 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.* . . . . . 178

# Referências

## Referências 1

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