

IN

DEV FULL STACK- PYTHON  
AULA 6 - HERANÇA

# Revisão



# Associação

- Uma classe com tipo de dados de uma outra classe
- Lê-se “Tem um”
- Define como as classes interagem entre elas.



# Associação

```
class Endereco:
    def __init__(self, cep, rua, numero, cidade, estado, pais, complemento=None):
        self.cep = cep
        self.rua = rua
        self.numero = numero
        self.cidade = cidade
        self.estado = estado
        self.pais = pais
        self.complemento = complemento
```

# Associação

```
class Cliente:
    def __init__(self, nome, email, senha, endereco):
        self.__nome = nome
        self.__email = email
        self.__senha = senha
        self.endereco = endereco
```

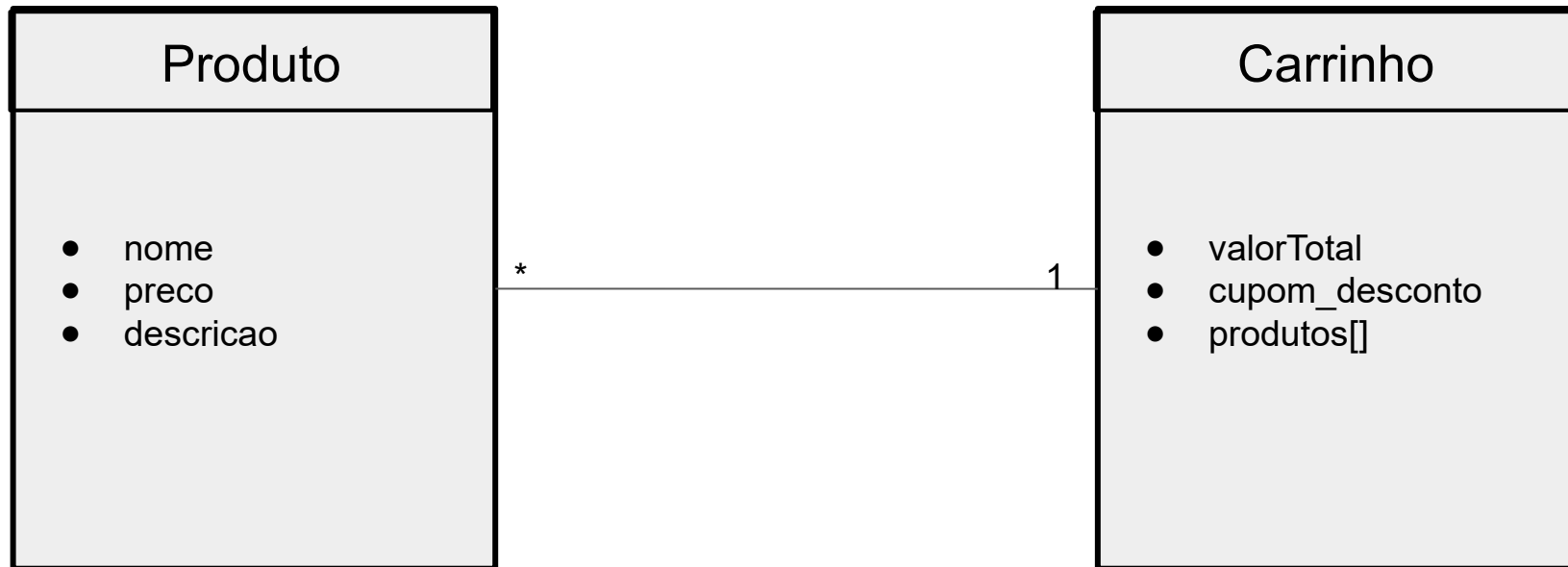
```
@property
def nome(self):
    return self.__nome
@nome.setter
def nome(self, nome):
    self.__nome = nome
@property
def email(self):
    return self.__email
@email.setter
def email(self, email):
    self.__email = email
```

```
@property
def senha(self):
    return self.__senha
@senha.setter
def senha(self, senha):
    self.__senha = senha
```

 main.py > ...

```
1  from Endereco import Endereco
2  from Cliente import Cliente
3  #DADOS DO CLIENTE
4  nome = input("Informe o seu nome: ")
5  email = input("Informe o seu email: ")
6  senha = input("Informe a sua senha: ")
7  #DADOS DO ENDEREÇO DO CLIENTE
8  cep = input("Informe o cep da rua: ")
9  rua = input("Informe a rua: ")
10 numero = input("Informe o número: ")
11 cidade = input("Informe a cidade: ")
12 estado = input("Informe o estado: ")
13 pais = input("Informe o país: ")
14 #Criando endereço
15 endereco1 = Endereco(cep, rua, numero, cidade, estado, pais)
16 #Criando um cliente
17 cliente1 = Cliente(nome, email, senha, endereco1)
18 print("Nome: ",cliente1.nome)
19 print("Email: ",cliente1.email)
20 print("Endereço: ",cliente1.endereco.rua, ", ", cliente1.endereco.numero)
```

# Agregação



# Agregação

```
class Produto:
    def __init__(self, nome, preco, descricao):
        self.__nome = nome
        self.__preco = preco
        self.__descricao = descricao

    @property
    def preco(self):
        return self.__preco

    @preco.setter
    def preco(self, preco):
        preco_min = self.__preco * 0.10;
        preco_min = self.__preco - preco_min;
        if preco_min >= self.__preco:
            self.__preco = preco
```

Carrinho.py > ...

```
1 class Carrinho:
2     def __init__(self, produtos):
3         self.__produtos = produtos
4
5     def valorTotal(self):
6         total = 0.0
7         for p in self.__produtos:
8             total += p.preco
9         return total
0
```



# Agregação

main.py > ...

```
1  from Produto import Produto
2  from Carrinho import Carrinho
3
4  produto1 = Produto("Escova", 30.5, "Escova de cabelo")
5  produto2 = Produto("Garrafa de água", 3.5, "Água mineral")
6  produto3 = Produto("Carteira", 80.0, "Carteira de couro")
7  produtos = [produto1, produto2, produto3]
8  carrinho = Carrinho(produtos)
9
10 print(carrinho.valorTotal())
11
12
```

# Atividade

- Desenvolva um sistema capaz de cadastrar alunos com nome, semestre e matrícula e esses alunos terão notas. As notas poderão ser cadastradas com um nome (ex: nota da prova 1), disciplina (ex: Lógica de programação) e valor (ex: 8.9). O aluno terá várias notas e ao final poderemos saber o score do aluno.

# Atividade

- Desenvolva um sistema para um processo seletivo. Serão capturados vários candidatos e eles farão uma prova que será a primeira etapa do processo. Todos os candidatos deverão ter informações de nome, endereço, tempo de experiência e descrição do candidato. A prova será feita em uma data específica e a pontuação que o candidato obteve. Teremos uma lista de candidatos aprovados, que são apenas os candidatos que tiveram uma nota superior a 8.

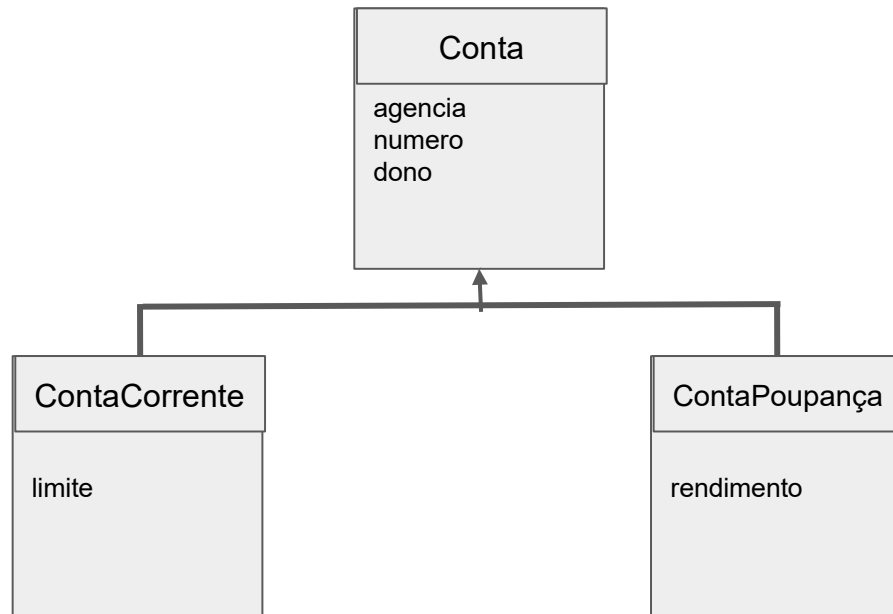
# Herança

—



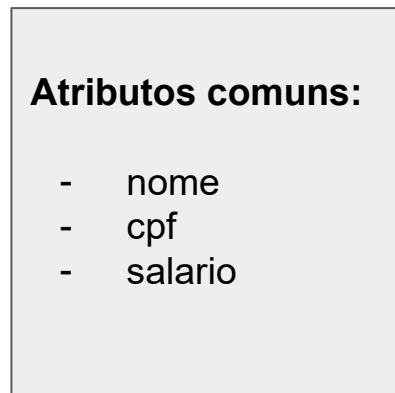
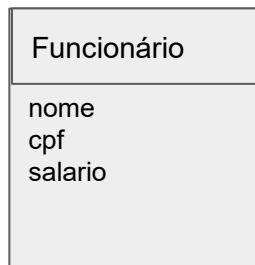
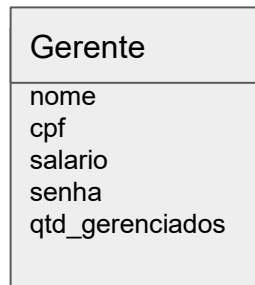
# Herança

- Conceito de Orientação a Objetos
- Uma classe pode herdar atributos e métodos de uma outra classe
- Classe filha x Classe mãe



# Situação

- Funcionário e Gerente



# Situação

- Funcionário e Gerente

Gerente
nome cpf salario senha qtd_gereciados

Funcionário
nome cpf salario

Funcionario.py > ...

```
1 class Funcionario:
2     def __init__(self, nome, cpf, salario):
3         self.__nome = nome
4         self.__cpf = cpf
5         self.__salario = salario
6
```

Gerente.py > ...

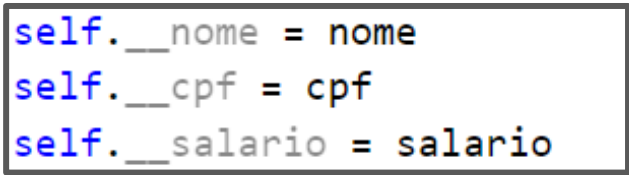
```
1 class Gerente:
2     def __init__(self, nome, cpf, salario, senha, qt_gereciados):
3         self.__nome = nome
4         self.__cpf = cpf
5         self.__salario = salario
6         self.__senha = senha
7         self.__qt_gereciados = qt_gereciados
8
```

# Situação

- Funcionário e Gerente

Gerente.py > ...

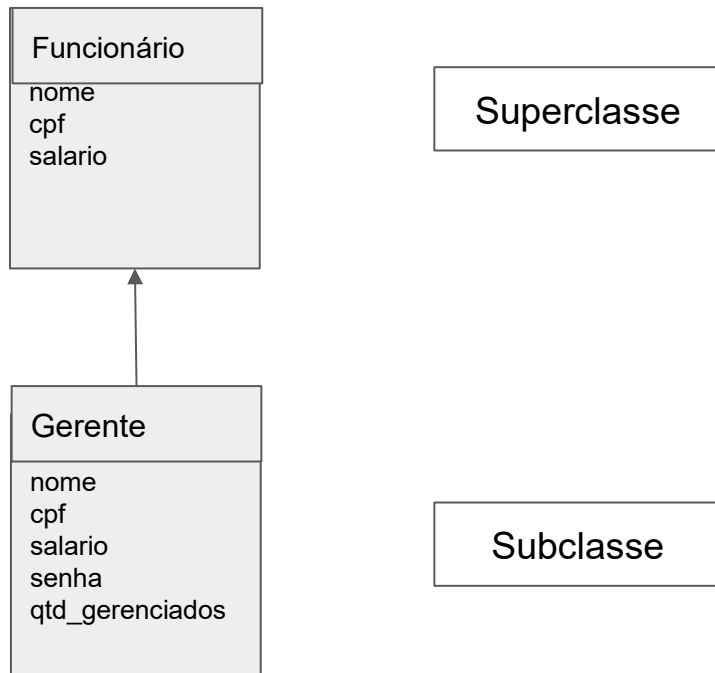
```
1  class Gerente:
2      def __init__(self, nome, cpf, salario, senha, qt_gerenciados):
3          self.__nome = nome
4          self.__cpf = cpf
5          self.__salario = salario
6          self.__senha = senha
7          self.__qt_gerenciados = qt_gerenciados
8
```



Repetindo...



# Situação - Resolvendo



# Situação - Resolvendo (no código)

```
class Gerente(Funcionario):  
    def __init__(self, nome, cpf, salario, senha, qt_gerenciados):  
        super().__init__(nome, cpf, salario)  
        self.__senha = senha  
        self.__qt_gerenciados = qt_gerenciados
```

# Herança

- Classe Gerente herda atributos e métodos da classe Funcionário

```
1  class Funcionario:
2      def __init__(self, nome, cpf, salario):
3          self._nome = nome
4          self._cpf = cpf
5          self._salario = salario
6
7      def salarioLiquido(self, itens_descontos):
8          liquido = self._salario
9          for desconto in itens_descontos:
10             liquido -= desconto
11         return liquido
```



# Herança

- Usa-se apenas um `_` para protected
- Protected: Classes filhas (subclasses) podem acessar os atributos da classe mãe(superclasse) como se fossem seus.

```
1  class Funcionario:
2      def __init__(self, nome, cpf, salario):
3          self._nome = nome
4          self._cpf = cpf
5          self._salario = salario
6
```

# Herança

main.py > ...

```
● 1  from Gerente import Gerente
  2
  3  gerente = Gerente("Ana", "4455577", 25000.00, "654321", 12)
  4  plano_saude = 300
  5  inss = 280
  6  i_r = 500
  7  lista_descontos = {plano_saude, inss, i_r}
  8  salario_liquido = gerente.salarioLiquido(lista_descontos)
  9  print(salario_liquido)
```

# Herança - Reescrita de Método

- Escrever especificamente qual o comportamento em uma classe específica

## Funcionario.py

```
def bonifica(self):  
    return self._salario * 0.1
```

## Gerente.py

```
def bonifica(self):  
    return self._salario * 0.15
```

# Herança - Reescrita de Método

main.py > ...

```
1  from Gerente import Gerente
2  from Funcionario import Funcionario
3
4  gerente = Gerente("Ana", "4455577", 25000.00, "654321", 12)
5  plano_saude = 300
6  inss = 280
7  i_r = 500
8  lista_descontos = {plano_saude, inss, i_r}
9  salario_liquido = gerente.salarioLiquido(lista_descontos)
10 print(salario_liquido)
11
12 print(gerente.bonifica())
```

**3.750,00**

# Exercício

Desenvolver um sistema capaz de cadastrar médicos com CRM, Nome, Idade e Salário. O médico pode ser aposentado, para isto, deveremos verificar no geral, se ele tem mais de 55 anos. Deveremos verificar também o valor da aposentadoria dele, que será normalmente 80% do valor do salário dele. Quando o médico é auxiliar ele só se aposentará com 60 anos. Diferente de se o médico for cirurgião, que será com 50. Quando o médico é cirurgião também, o valor da aposentadoria dele tem, além dos 80% do salário um acréscimo de 2000 reais.



# Exercício



Desenvolver um sistema que auxilie no cadastro de instrumentos musicais de uma escola de música. Os instrumentos devem ser cadastrados com nome, grau de dificuldade e o professor que ensina o instrumento. O professor será cadastrado no sistema com nome e pontuação (de 1 a 10). Os instrumentos serão divididos por tipo, pois eles podem ser instrumento de corda, sopro ou percussão. Quando o instrumento é de corda, pode-se inserir informações da quantidade de cordas e tipo de corda. Já quando o instrumento é de percussão, pode ser informado se usa baqueta ou não. O grau de dificuldade do instrumento é calculado em primeiro lugar pela pontuação do professor que a leciona. Mas se o instrumento for de corda, se a corda for de aço, é multiplicado por quantidade de cordas pelo grau de dificuldade.

The logo consists of the letters 'IN' in a white, serif font, centered within a solid red square. The background of the entire image is a dark red with abstract, layered geometric shapes in various shades of red, creating a sense of depth and movement.

# IN

71 3901 1052 | 71 9 9204  
0134

@infinity.school

[www.infinityschool.com.br](http://www.infinityschool.com.br)

Salvador Shopping Business | Torre Europa Sala  
310 Caminho das Árvores, Salvador - BA CEP: