



IN

dev full stack- Python

Aula 4 - Python OO

Orientação a Objetos

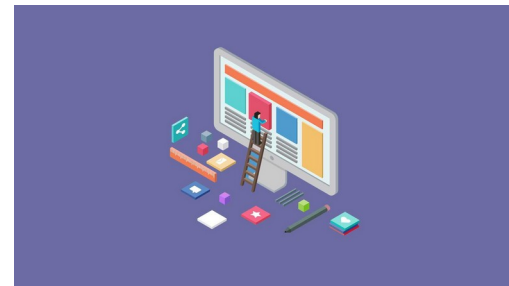
Classe



MinhaClasse.py

Classes

- Forma de definir um tipo de dado;
- Formada por dados e comportamentos;
- Os dados são chamados de Atributos e são as variáveis que representam um item do mundo real;



Classes

Exemplos:

Carro
marca modelo ano automatico
acelerar() frear() ligar()

ContaCorrente
numero agencia banco senha saldo
depositar() sacar() transferir()

Aluno
nome endereco idade cpf
matricular() estudarModulo()

Classes - Construtor e Atributos

- **Construtor:**
 - Na criação do objeto obrigará atribuição de valores iniciais para atributos;
- **Atributos:**
 - Características importantes para a classe

ContaCorrente
numero agencia banco senha saldo
depositar() sacar() transferir()

Classes - Construtor e Atributos

- **Construtor**
- **Atributos**

```
class ContaCorrente:  
    def __init__(self, numero, senha, agencia='555', banco='01', saldo=15.0):  
        self.numero = numero  
        self.senha = senha  
        self.agencia = agencia  
        self.banco = banco  
        self.saldo = saldo
```

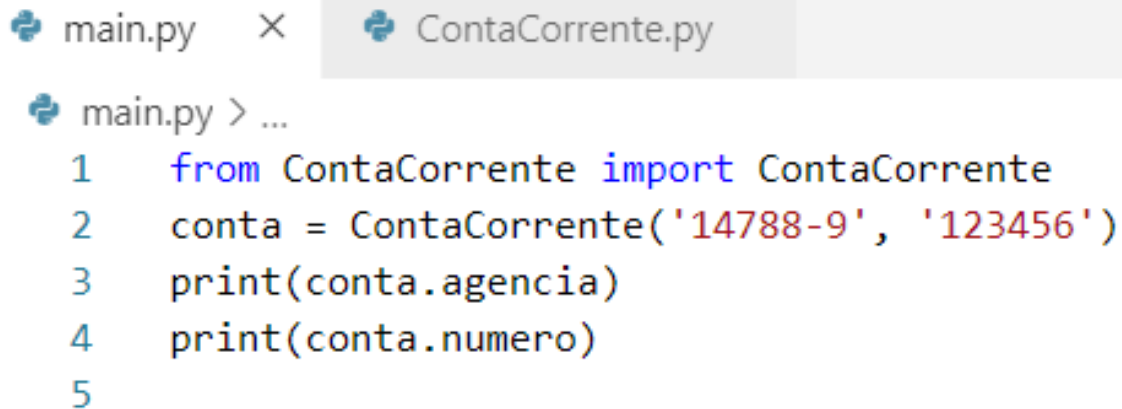
Construtor

ContaCorrente

numero
agencia
banco
senha
saldo

depositar()
sacar()
transferir()

Objetos



```
main.py  X  ContaCorrente.py  
main.py > ...  
1  from ContaCorrente import ContaCorrente  
2  conta = ContaCorrente('14788-9', '123456')  
3  print(conta.agencia)  
4  print(conta.numero)  
5
```

Atribuição de valores
Item do mundo real

Self

```
from ContaCorrente import ContaCorrente
```


```
conta = ContaCorrente('14788-9', '123456')  
print(conta.agencia)  
print(conta.numero)
```

Objeto 1

```
conta2 = ContaCorrente('14733-6', '654321')  
print(conta.agencia)  
print(conta.numero)  
print(conta.saldo)
```

Objeto 2

Self



```
from ContaCorrente import ContaCorrente

{conta} = ContaCorrente('14788-9', '123456')
print(conta.agencia)
print(conta.numero)

conta2 = ContaCorrente('14733-6', '654321',
print(conta.agencia)
print(conta.numero)
print(conta.saldo)

class ContaCorrente:
    def __init__(self, numero, senha, agencia, banco, saldo):
        self.numero = numero
        self.senha = senha
        self.agencia = agencia
        self.banco = banco
        self.saldo = saldo
```

Métodos

- Comportamentos que o objeto pode ter;
- Modifica os valores dos atributos do objeto;

Carro
marca modelo ano automatico
acelerar() frear() ligar()

ContaCorrente
numero agencia banco senha saldo
depositar() sacar() transferir()

Aluno
nome endereco idade cpf
matricular() estudarModulo()

Métodos

```
class ContaCorrente:
```

```
    def __init__(self, numero, senha, agencia='555', banco='01', saldo=15.0):  
        self.numero = numero  
        self.senha = senha  
        self.agencia = agencia  
        self.banco = banco  
        self.saldo = saldo
```

```
from ContaCorrente import ContaCorrente  
conta = ContaCorrente('14788-9', '123456')  
print(conta.agencia)  
print(conta.numero)
```

```
conta2 = ContaCorrente('14733-6', '654321')  
print(conta.agencia)  
print(conta.numero)  
print(conta.saldo)
```

```
conta.sacar(10)
```

```
print(conta.saldo)
```

```
def sacar(self, valor):  
    self.saldo = self.saldo - valor
```

Inserir condicional

Atividade

Criar uma classe de Carro, que corre até no máximo 120km/h. O carro deve ser cadastrado com marca, modelo, ano, velocidade, se está ligado ou não e se é automático ou não.

O carro deve conter funcionalidades de:

- ligar
- Acelerar: apenas se o carro estiver ligado (uma quantidade que não passe da velocidade máxima de 120.
- desligar
- verificarMacha com as regras abaixo:
- **1ª marcha:** 0 a 20km
- **2ª marcha:** ao atingir 30 km/h;
- **3ª marcha:** entre 30 e 35 km/h;
- **4ª marcha:** entre 35 e 50 km/h; e.
- **5ª marcha:** acima de 50 km/h.

Orientação a Objetos



Encapsulamento



Encapsulamento

- Proteger os atributos da nossa classe;
- Inserir regras nas atribuições de valores;

Encapsulamento - Análise

Produto.py > ...

```
1 class Produto:
2     def __init__(self, nome, preco, descricao):
3         self.nome = nome
4         self.preco = preco
5         self.descricao = descricao
6     #preços podem ter até 10% de desconto
```

main.py > ...

```
1 from Produto import Produto
2
3 produto = Produto("Escova", 30.5, "Escova de cabelo")
4 produto.preco = 10.0
5
6 print("Produto: ", produto.nome, "\nValor: ", produto.preco)
```


Encapsulamento

- Como criamos regras para classe???

MÉTODOS!

GET e SET

- É possível criar métodos para recuperar valores de atributos e atribuir novos valores a eles. Isso permite que as aplicações manipulem os atributos sem fazer acesso direto aos mesmos.
- A grande vantagem disso é que o atributo pode ser “protegido” do acesso direto, já que toda manipulação a ele acontece por meio de um método que pode ser programado de acordo com o que o projetista da classe deseja.
 - } GET é como chamamos o método de recuperação do atributo.
 - } SET é como chamamos o método de atribuição de valor ao atributo.

Em nosso exemplo da classe Produto...

class Produto:

```
def __init__(self, nome, valor, descricao):
```

```
    self.nome = nome
```

```
    self.__preco = valor
```

```
    self.descricao = descricao
```

preços podem ter até 10% de desconto na mudança de valor

```
@property
```

```
def preco(self):
```

```
    return self.__preco
```

GET

Esta solução é considerada uma boa prática de encapsulamento, principalmente se houver algum tipo de lógica a ser incluída nos processos de recuperação ou alteração de valor do atributo.

```
@preco.setter
```

```
def preco(self, val):
```

```
    preco_min = self.__preco * 0.10
```

```
    preco_min = self.__preco - preco_min # mínimo é 10% menos que o preço original
```

```
    if (val >= preco_min):
```

```
        self.__preco = val
```

```
        return True
```

```
    else:
```

```
        return False
```

SET

Usando a solução de Encapsulamento

class Produto:

```
def __init__(self, nome, valor, descricao):
```

```
    self.nome = nome
```

```
    self.__preco = valor
```

```
    self.descricao = descricao
```

preços podem ter até 10% de desconto na
mudança de valor

@property

```
def preco(self):
```

```
    return self.__preco
```

@preco.setter

```
def preco(self, val):
```

```
    preco_min = self.__preco * 0.10
```

```
    preco_min = self.__preco - preco_min
```

```
    if (val >= preco_min):
```

```
        self.__preco = val
```

```
        return True
```

```
    else:
```

```
        return False
```

main.py:

```
from Produto import Produto
```

```
produto1 = Produto('Escova', 40.0, 'Escova de cabelo')
```

```
print ('-----')
```

```
print ('Valor do Produto (antes):', produto1.preco)
```

```
produto1.preco = 38.0
```

```
print ('-----')
```

```
print ('Valor do Produto (depois):', produto1.preco)
```

```
print ('-----')
```

Resultado:

```
-----
```

```
Valor do Produto (antes): 40.0
```

```
-----
```

```
Valor do Produto (depois): 38.0
```

```
-----
```

Outra forma de fazer...

```
class Produto:
    def __init__(self, nome, valor, descricao):
        self.nome = nome
        self.preco = valor
        self.descricao = descricao
# preços podem ter até 10% de desconto na mudança de valor
```

```
def getPreco(self):
    return self.preco
```

GET

```
def setPreco(self, val):
    preco_min = self.preco * 0.10
    preco_min = self.preco - preco_min # mínimo é 10% menos que o preço original
    if (val >= preco_min):
        self.preco = val
        return True
    else:
        return False
```

SET

Esta solução é considerada uma maneira menos elegante de resolver o problema de encapsulamento, mas também funciona.

Usando a 2ª solução de Encapsulamento

class Produto:

```
def __init__(self, nome, valor, descricao):
```

```
    self.nome = nome
```

```
    self.preco = valor
```

```
    self.descricao = descricao
```

preços podem ter até 10% de desconto na mudança de valor

```
def getPreco(self):
```

```
    return self.preco
```

```
def setPreco(self, val):
```

```
    preco_min = self.__preco * 0.10
```

```
    preco_min = self.__preco - preco_min # mínimo é 10%
```

menos que o preço original

```
    if (val >= preco_min):
```

```
        self.__preco = val
```

```
        return True
```

```
    else:
```

```
        return False
```

main.py:

```
from Produto import Produto
```

```
produto1 = Produto('Escova', 40.0, 'Escova de cabelo')
```

```
print ('-----')
```

```
print ('Valor do Produto (antes):', produto1.getPreco())
```

```
produto1.setPreco(38.0)
```

```
print ('-----')
```

```
print ('Valor do Produto (depois):', produto1.getPreco())
```

```
print ('-----')
```

Resultado:

```
-----
```

```
Valor do Produto (antes): 40.0
```

```
-----
```

```
Valor do Produto (depois): 38.0
```

```
-----
```

Atividade

- Desenvolva um sistema onde possa ser cadastrado dados de um funcionário que terá nome, salário, matrícula e função. Os valores serão cadastrados pelo RH no momento de criação do cadastro. Porém depois de um tempo, será possível alterar o salário do funcionário em até 20% a mais. E nunca será possível alterar o salário para um valor menor.

The logo consists of the letters 'IN' in a white, serif font, centered within a solid red square. The background of the entire slide is a solid red color with abstract, darker red geometric shapes and lines creating a sense of depth and movement.

71 3901 1052 | 71 9 9204
0134

@infinity.school

www.infinityschool.com.br

Salvador Shopping Business | Torre Europa Sala 310
Caminho das Árvore, Salvador - BA CEP: 40301-155