

**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**  
RIO GRANDE DO NORTE



# Programação Orientada a Objetos

## APRESENTAÇÃO DA DISCIPLINA

**PROFESSOR:**

Demetrios Coutinho

[Demetrios.coutinho@ifrn.edu.br](mailto:Demetrios.coutinho@ifrn.edu.br)

# AGENDA

- Conceitos Básicos de POO
  - Abstração
  - Definição de atributos, classes e instâncias
  - Construtores

## O PARADIGMA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

- Paradigmas são formas de enxergar o mundo (os problemas, a vida, um código de um programa)
- O paradigma de **Orientação a Objetos** pode ser encarado como uma forma de pensar o seu projeto, desde a arquitetura até a implementação.
- Outros exemplos de paradigmas de programação são a **programação imperativa** e a **orientada a procedimentos**

# O PARADIGMA ORIENTADO A PROCEDIMENTOS

- Baseado em chamada de funções ou sub-rotinas que operam sobre elas.
- O fluxo de dados concentra todas as variáveis
- Uma função toma um conjunto de variáveis como argumento e retorna o resultado para o fluxo de dados, para ser usado por outra função ou simplesmente ser exibido para o usuário.



```
def cria_conta(numero, titular, saldo, limite):  
    conta = {"numero": numero, "titular": titular, "saldo": saldo, "limite": limite}  
    return conta
```

Red handwritten annotations: a bracket on the left of the function definition, a red arrow pointing to the function name 'cria\_conta', and red dashed lines above the function arguments.

# PARADIGMA ORIENTADO A OBJETOS

- Estamos rodeados por objetos: mesa, carro, livro, pessoa, etc; e
- Os objetos do mundo real têm duas características em comum:
  - Estado = propriedades (nome, peso, altura, cor, etc.);
  - Comportamento = ações (andar, falar, calcular, etc.).

Bancos

Propriedades: Nome, Juros de em,  
Segurança, atendente  
Ação: Pagar, sacar dinheiro,  
Deposito, investir.

Pessoa

Conta

Propriedades: tipo,  
agencia, Titular, saldo.  
Ação: Pix, transferência,  
Ver status.

- POO para desenvolvimento de software baseia-se na **utilização de componentes individuais (objetos)** que colaboram para construir sistemas mais complexos.

# PARADIGMA ORIENTADO A OBJETOS

## VANTAGENS

- Facilita a **reutilização** de código;
- Os modelos refletem o mundo real de maneira mais aproximada:
  - Descrevem de maneira mais precisa os dados;
  - Mais fáceis de entender e manter.
- Pequenas mudanças nos requisitos **não implicam em grandes alterações** no sistema em desenvolvimento.



# OS QUATRO PILARES

ABSTRAÇÃO

ENCAPSULAMENTO

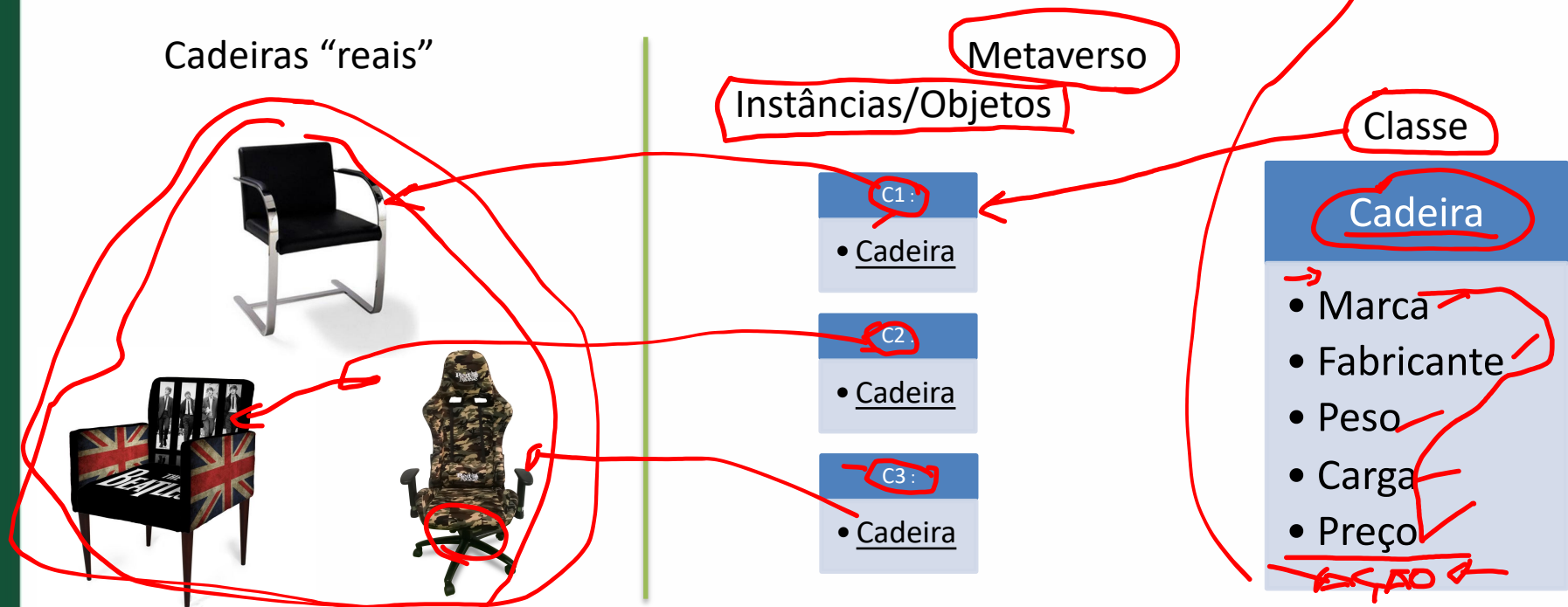
HERANÇA

POLIMORFISMO

# ABSTRAÇÃO

- A estrutura fundamental para definir novos objetos;
- Uma classe é definida em código-fonte.

Def Girar( graus):  
azimute += graus





# ABSTRAÇÃO

## CLASSES EM PYTHON

Estrutura:

**class** nome\_\_da\_\_Classe:

-> -> atributos

-> -> métodos

```
class Cadeira:
    modelo = 'Racing Series'
    Fabricante = 'DT3sports'
    Peso = 28
    Carga = 180
    Preco = 2000
```

↳ ATRIBUTO

# ABSTRAÇÃO

## INSTÂNCIA

- Uma instância é um objeto criado com base em uma classe definida;
- Classe é apenas uma estrutura, que especifica objetos, mas que não pode ser utilizada diretamente;
- Instância representa o objeto concretizado a partir de uma classe;
- Uma instância possui um ciclo de vida:





# ABSTRAÇÃO

## INSTÂNCIA EM PYTHON

Estrutura:

variável = Classe()

```
cadeira_1 = Cadeira()  
cadeira_1.preco = 2500
```

# MÉTODOS

- Representam os comportamentos de uma classe;
- Permitem acesso a atributos, tanto para recuperar os valores, como para alterá-los caso necessário;
- Podem retornar ou não algum valor; e
- Podem possuir ou não parâmetros.

# MÉTODOS

## Estrutura:

def nome\_do\_metodo(self parametros)

- **Importante:** o parâmetro **self** é obrigatório.
- Esse argumento **self** é simplesmente uma referência ao objeto.

```
def saca(self, valor):  
    if (self.saldo < valor):  
        return False  
    else:  
        self.saldo -= valor  
        return True
```

Método com retorno

```
class Conta:  
    numero = "00000-0"  
    saldo = 0.0  
  
    def deposito(self, valor):  
        self.saldo += valor  
  
    def saque(self, valor):  
        if (self.saldo > 0):  
            self.saldo -= valor  
        else:  
            print("Saldo insuficiente")
```

# MÉTODO CONSTRUTOR

- Determina que ações devem ser executadas na criação de um objeto; e
- Pode possuir ou não parâmetros.

Estrutura:

def \_\_init\_\_(self, parametros)

```
class Conta:
    numero = "00000-0"
    saldo = 0.0

    def __init__(self, numero, saldoInicial):
        self.numero = numero
        self.saldo = saldoInicial

conta = Conta("12345-1", 0)
print(conta.numero)
print(conta.saldo)
```

# Objetos são acessados por referência

- A instância:

```
c1 = Conta()
```

O correto é dizer que c1 se refere a um objeto. Não é correto dizer que c1 é um objeto, pois c1 é uma variável referência. Todavia, é comum ouvir frases como “tenho um objeto c1 do tipo Conta”, mas isso é apenas uma abreviação para encurtar a frase “Tenho uma referência c1 a um objeto tipo Conta”.

# Objetos são acessados por referência

```
class Conta:

    def __init__(self, numero, titular, saldo, limite):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite

    def deposita(self, valor):
        self.saldo += valor

    def saca(self, valor):
        self.saldo -= valor

    def extrato(self):
        print("numero: {} \nsaldo: {}".format(self.numero, self.saldo))
```



# Objetos são acessados por referência

```
c1 = Conta('123-4', 'João', 120.0, 1000.0)
```

```
c2 = c1
```

```
print(c2.saldo)
```

```
#120.0
```

```
c1.deposita(100.0)
```

```
print(c1.saldo)
```

```
#220.0
```

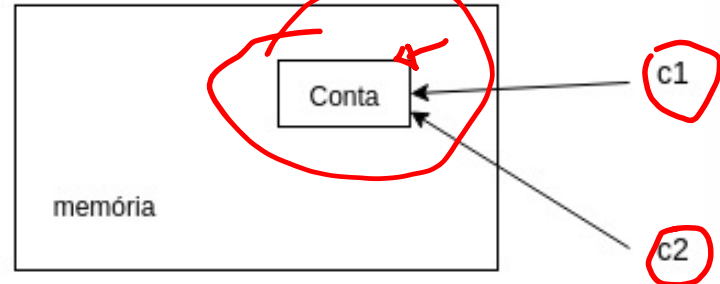
```
c2.deposita(30.0)
```

```
print(c2.saldo)
```

```
#250.0
```

```
print(c1.saldo)
```

```
#250.0
```



```
print(id(c1))
```

```
#140059774918104
```

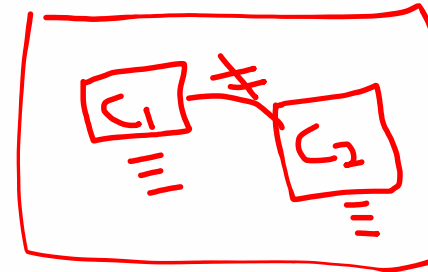
```
print(id(c2))
```

```
#140059774918104
```

- Ao fazer `c2 = c1`, `c2` passa a fazer referência para o mesmo objeto que `c1` referencia nesse instante.
- Para quem conhece, é parecido com um ponteiro, porém você não pode manipulá-lo.

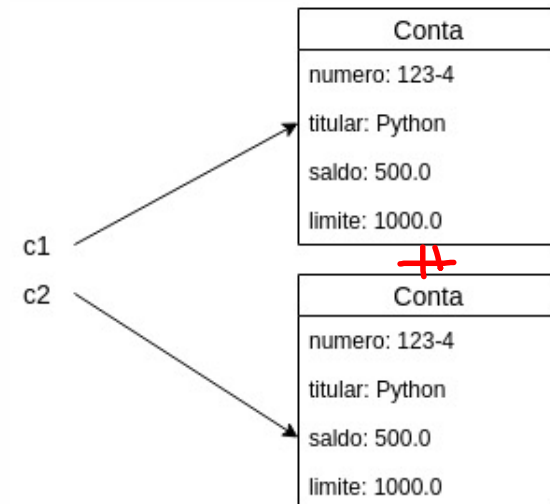
# Objetos são acessados por referência

```
c1 = Conta("123-4", "Python", 500.0, 1000.0)
c2 = Conta("123-4", "Python", 500.0, 1000.0)
if(c1 == c2):
    print("contas iguais")
```



**FALSE**

As contas podem ser equivalentes no  
nosso critério de igualdade, porém elas  
não são o mesmo objeto.



# Tudo em Python é um objeto!

- Python é uma linguagem totalmente orientada a objetos.
- Sempre que utilizamos uma função ou método que recebe parâmetros, estamos passando objetos como argumentos.
- O mesmo acontece com numero, saldo e limite. Strings e números são classes no Python.

```
print(type(conta.numero))  
#<class 'str'>  
print(type(conta.saldo))  
#<class 'float'>  
print(type(conta.titular))  
#<class '__conta__Cliente'>
```

# EXERCÍCIO

Classe Pessoa: Crie uma classe que modele uma pessoa:

- **Atributos**: nome, idade, peso e altura
- **Métodos**: Envelhecer, engordar, emagrecer, crescer.
- **Obs**: Por padrão, a cada ano que a pessoa envelhece, sendo a idade dela menor que 21 anos, ela deve crescer 0,5 cm.
- O construtor pode possuir ou não parâmetros.

# EXERCÍCIO

class **Conta**:

```
def __init__(self, numero, titular, saldo, limite):
    self.numero = numero
    self.titular = titular
    self.saldo = saldo
    self.limite = limite

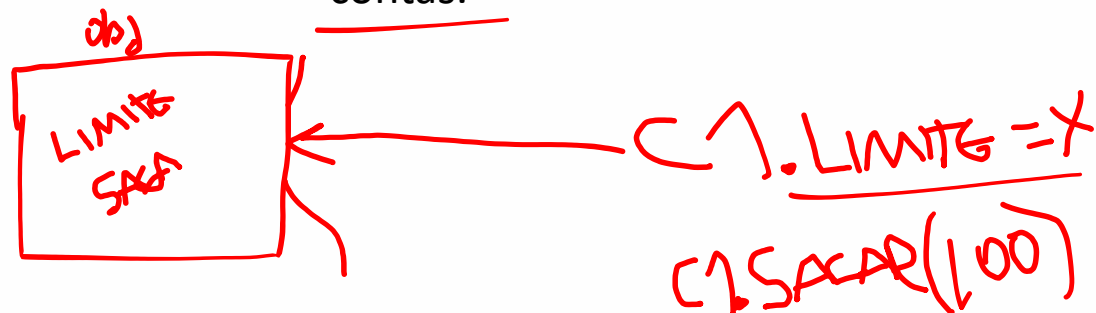
def deposita(self, valor):
    self.saldo += valor

def saca(self, valor):
    self.saldo -= valor

def extrato(self):
    print("numero: {} \nsaldo: {}".format(self.numero, self.saldo))
```

- Altere o método **saca** para retornar **True**, caso tenha saldo e **Falso**, caso contrário.
- Crie um método transferir um valor para um destino.
- Acrescente o atributo codigo\_tipo e nome\_tipo. 01 para "conta corrente" e 02 para "poupança".

Faça um programa que receba as informações do terminal para criar duas contas e que faça uma transferência entre contas.



# DÚVIDAS?

