Estrutura de dados
Orientação a Objetos

Profª. Drª. Alana Morais

Finalmente!!

Chegamos ao mundo OO

Introdução

- O que é um paradigma?
- Quais paradigmas conhecemos?
- Por que temos que amadurecer e conhecer o paradigma OO?
- Onde programaremos?

Orientação a Objetos

- Atualmente
  - Escrevemos funções seguindo o paradigma Procedural.
  - Criamos funções, que recebem parâmetros, realizam algo e retornam um valor.
  - Utilizamos a programação procedural para escrever nossos programas — e tem funcionado muito bem.
  - Aprendemos as principais técnicas de Estrutura de Dados.

Orientação a Objetos - Exemplo Inicial

- Imagine que você tem uma conta e ela possuirá algumas características: saldo, número, agência, titular e um limite.

  numero = 123
  titular = "Nico"
  saldo = 55.0
  limite = 1000.0

Orientação a Objetos - Exemplo Inicial

- E se agora quiséssemos representar um sistema com diversas contas.

```
numero = 123
titular = "Nico"
saldo = 55.0
limite = 1000.0

numero2 = 124
titular2 = "Manoel"
saldo2 = 50.0
limite2= 100.0
```

Orientação a Objetos - Exemplo Inicial

- E se agora quiséssemos representar um sistema com diversas contas.

```
//ou dicionários
conta1 = {"numero": 123, "titular": "Nico", "saldo": 55.0, "limite": 1000.0}
conta2 = {"numero": 124, "titular": "Manoel", "saldo": 50.0, "limite": 100.0}
```

- Eu conseguiria criar uma função para criar automaticamente esses dicionários?
- Como eu acesso tais características para modificar os campos do dicionário? Por exemplo, sacar ou depositar.

Orientação a Objetos - Exemplo Inicial

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

def depositar(conta, valor):
        conta["saldo"] += valor

def sacar(conta, valor):
        conta["saldo"] -= valor

def imprimir_extrato(conta):
        return ("Saldo {}".format(conta["saldo"]))
```

Orientação a Objetos - Exemplo Inicial

```
def main():
        conta = cria_conta(123, "Nico", 55.0, 1000.0)
        deposita(conta, 300.0)
        print(extrato(conta)) #Saldo 355.0
        saca(conta, 100.0)
        print(extrato(conta)) #Saldo 255.0

main()
```

Por que esse modo de programar pode ser melhorado?

- Organização fica restrita ao desenvolvedor
  - Em um sistema maior, é grande a chance de que as funções fiquem separadas em arquivos e módulos diferentes do projeto.
- Nada impede que as informações sejam acessadas diretamente
- Trabalhar em um projeto em andamento pode ser um pesadelo
- Mudanças no processo são temidas

Orientação a Objetos

- O paradigma da Orientação a Objetos está relacionado com a organização do código e reuso
- A ideia central é enxergar os problemas de uma forma mais alto nível (o que chamamos de abstração)
- Conceitos importantes no entendimento sobre OO:
  - Classe
    - Atributos
    - Métodos
  - Construtor
  - Objeto
  - Instância

Orientação a Objetos - Classe

- Uma classe é um gabarito para a definição dos objetos, definido pelo programador
- As classes não ocupam espaço na memória por serem abstrações
- Pode ou não conter um método main
- Composta geralmente por:
    - Nome da classe: identificador para a classe, que permite referenciá-la
    - Atributos: descrevem características da classe
    - Métodos: definem as funcionalidades da classe

Orientação a Objetos - Classe

- Sintaxe:

```
class NomeClasse:
        construtor
        métodos
```

- As classes devem ter nomes iniciados com letras maiúsculas
- Métodos e atributos devem ser iniciadas com letras minúsculas
- Em Python, alguns nomes de métodos estão reservados para o uso da própria linguagem

Orientação a Objetos - Atributos

- O conjunto de atributos descreve as propriedades da classe
- É possível ter dados primitivos e objetos (falaremos mais sobre isto)
    - Dados de tipos primitivos são sempre referenciados por valor
    - Os objetos são sempre são sempre referenciados por meio de sua referência.
- O atributo pode ainda ter um valor *default* opcional.

Orientação a Objetos - Métodos

- Representam os comportamentos de uma classe
- Permitem que acessemos os atributos, tanto para recuperar os valores, como para alterá-los caso necessário
- Podem retornar ou não algum valor
- Podem possuir ou não parâmetros
- O parâmetro self é obrigatório
- Sintaxe:

```
def nome_do_método(self, parâmetros):
    #podem ou não ter return
```

Exemplo Inicial

Vamos planejar então quais seriam os atributos e os métodos para tornar o exemplo da conta uma classe.

Quais seriam os atributos?

Quais seriam os métodos?

```
numero = 123
titular = "Nico"
saldo = 55.0
limite = 1000.0
```

Exemplo Inicial

Vamos planejar então quais seriam os atributos e os métodos para tornar o exemplo da conta uma classe.

Quais seriam os atributos?

numero, titular, saldo, limite

Quais seriam os métodos?

Exemplo Inicial

Vamos planejar então quais seriam os atributos e os métodos para tornar o exemplo da conta uma classe.

Quais seriam os atributos?

numero, titular, saldo, limite

Quais seriam os métodos?

criar conta, depositar, sacar, imprimir extrato

Orientação a Objetos - Construtor

- Determina que ações devem ser executadas quando da criação de um objeto
- Método __init__ em Python
- Seu primeiro parâmetro, assim como todo método de instância, é a própria instância.
    - Por convenção, chamamos este argumento de **self**.
    - Não possui retorno

- Sintaxe:

```
def __init__(self, parâmetros):
```

Orientação a Objetos - Construtor

- Determina que ações devem ser executadas quando da criação de um objeto
- Sintaxe:

```
class Conta:
    def __init__(self, numero):
        self.numero = numero
        self.saldo = 0.0


conta = Conta (1)
print(conta.numero)
print(conta.saldo)
```

Orientação a Objetos - Construtor

- Apesar de muitos programadores chamarem o init de construtor, ele não cria sozinho um objeto.
- Existe outro método, o __new__() que é chamado antes do __init_() pelo interpretador do Python.
- O método __new__() é realmente o construtor e é quem realmente cria uma instância de um objeto.
- O método __init__() é responsável por inicializar o objeto, tanto é que já recebe a própria instância (self) criada pelo construtor como argumento.

Exemplo Inicial

Vamos reestruturar a classe Conta.py como estamos planejando.

Atributos e métodos

Como eu leio e modifico os valores dos atributos?

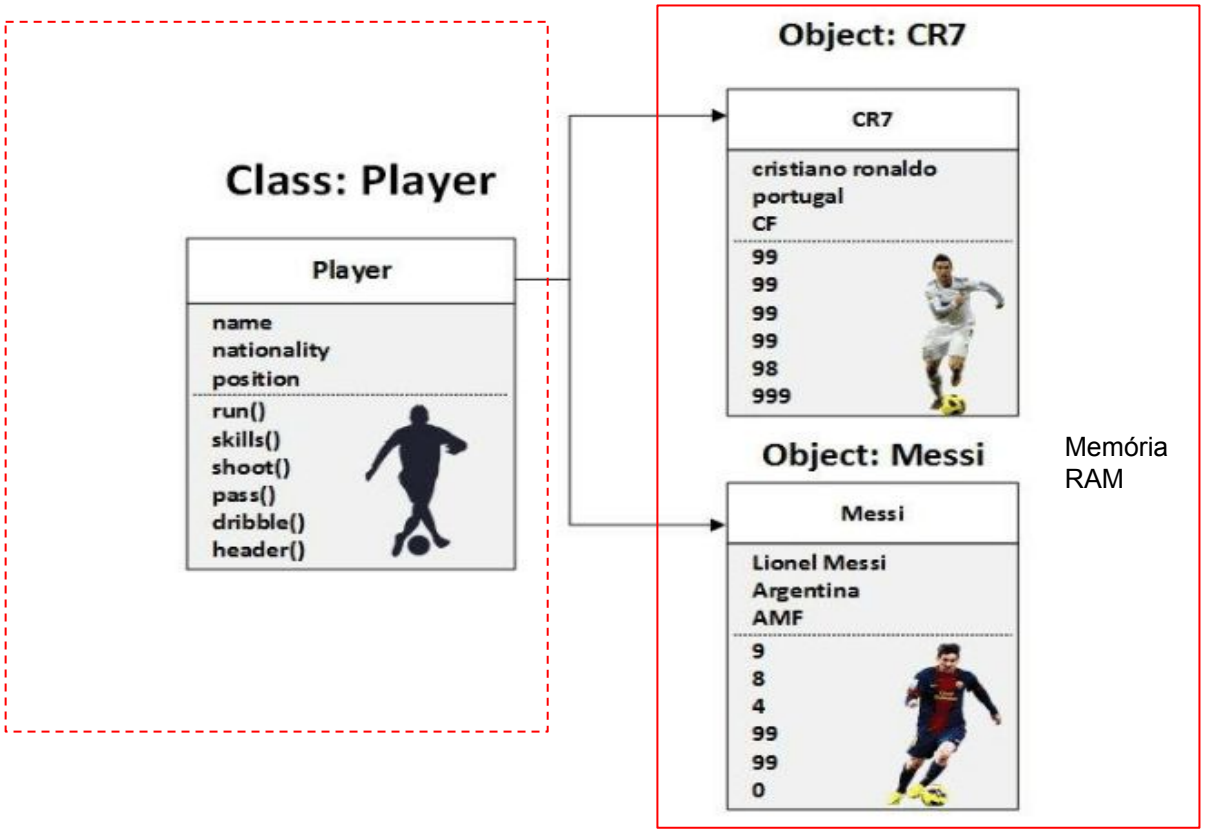
Como eu executo os métodos?

Orientação a Objetos - Objeto

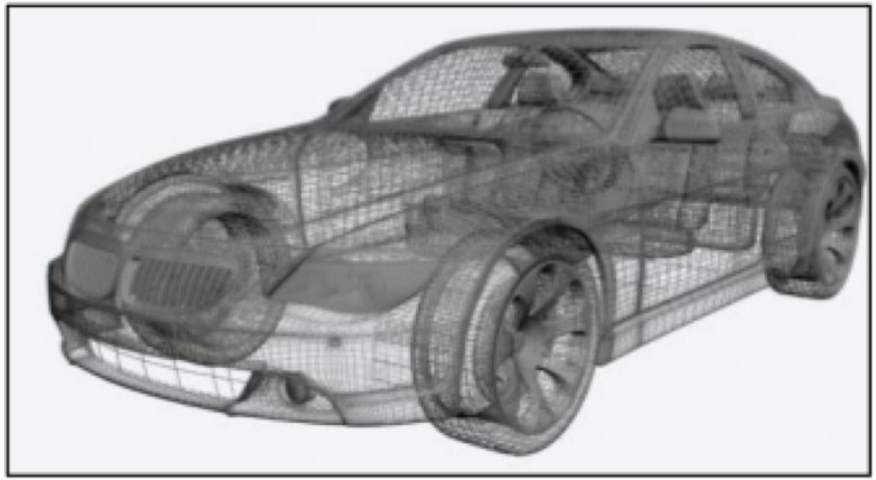
- É uma representação computacional de uma entidade do mundo real
- Uma particular instância de uma classe é chamada objeto
- Comparamos as classes às fábricas e os objetos aos produtos feitos por elas.
- Sintaxe:

```
from classeArquivo import NomeClasse

variavel = NomeClasse()

variavel.atributo = 10

variavel.metodo()
```

Orientação a Objetos - Objeto



Orientação a Objetos - Objeto



Classe

Objeto

Orientação a Objetos - Objeto

- Assim como as coisas no mundo real, os objetos tem “estado” e “comportamento”
  - Estado são informações sobre o objeto, como a sua cor, seu peso, o saldo da conta-corrente, etc.
  - Comportamento são coisas que podem ser feitas com ou pelo objeto, como depositar em uma conta-corrente ou mudar a cor de uma janela

- Cuidado: declarar é diferente de instanciar (para outras linguagens)

Exemplo Inicial

Crie dois objetos do tipo Conta chamado c1 e c2.
Inicialize tais objetos com os mesmos valores.

Depois, escreva um método que verifique se os objetos são iguais ou não.

Orientação a Objetos - Instância

- Uma instância é um objeto criado com base em uma classe definida
- Instância representa o objeto concretizado a partir de uma classe
- Cada instância possui o seu próprio conjunto de atributos, independente de outras instâncias da mesma ou de outras classes
- Todas as instâncias de uma mesma classe compartilham as mesmas definições de métodos

Exercício

Quais os problemas deste código?

```
class Pessoa:
    def __init__(self, nome, sobrenome):
        self.nome = nome
        self.sobrenome = sobrenome

    exibe_nome_e_sobrenome():
        print("{0} {1}".format(self.nome, self.sobrenome))


pessoa = Pessoa("Chalita", "Steppat")
pessoa.exibe_nome_e_sobrenome()
```

Ciclo de vida de um objeto

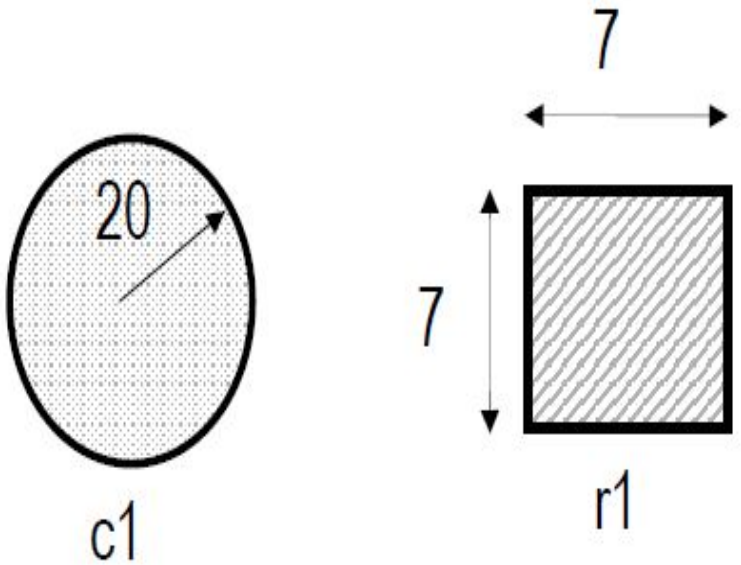
- Instanciação: o objeto é criado na memória e passa a ser referenciado por uma variável de referência;
- Uso: o objeto recebe mensagens de outros objetos e, com isso, executa parte da funcionalidade do sistema;
- Destruição: quando o objeto não é mais referenciado (inacessível) ele torna-se elegível para a coleta de lixo.

Planejamento OO

1. Identifique uma classe que deve existir no seu problema
    a. Pense nos atributos
    b. Implemente o construtor
    c. Implemente métodos adicionais
2. Verifique se há outras classes
    a. Repita o processo
3. Implemente o arquivo teste.py

Exercício

Implemente um programa que calcule a área e o perímetro de Retângulos e Círculos. Como ilustrado na figura abaixo:



Exercício Casa

1. Crie uma classe Aluno com nome, matricula, endereço e cpf. Teste e exiba a informação dos alunos por meio do método chamado infoAlunos().
2. Modele um funcionário. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário (double), a data de entrada no banco (String) e seu RG (String). Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método recebeAumento que aumenta o salário do funcionário de acordo com o parâmetro passado como argumento. Crie também um método calculaGanhoAnual, que não recebe parâmetro algum, devolvendo o valor do salário multiplicado por 12. Por fim, crie uma aplicação teste para verificar sua classe.

Dúvidas?

alanamm.prof@gmail.com