Classes e objetos

1. Classes

1.1. Criando uma classe

As classes são uma espécie de modelo para os objetos que serão criados. Nela definiremos todas as características (atributos, representados por variáveis internas) e métodos (ações, representadas por funções) de nossos objetos.

Para criar uma classe em Python, utilizamos a palavra class seguida do nome que daremos à classe. Tudo o que vier indentado a seguir fará parte da classe.

class MinhaClasse:

# código da classe aqui

As regras para nomes válidos de classe são as mesmas para variáveis e funções, mas um padrão muito utilizado é: Classe começa com letra maiúscula, objeto começa com letra minúscula. Especificamente no caso de classes, não é recomendável separar palavras por \_. Apenas coloque a primeira letra de cada palavra em maiúscula. Esse padrão é conhecido como Camel Case.

Por hora, tudo o que criaremos dentro de nossas classes serão os nossos métodos.

1.2. Métodos

Métodos são funções que representam ações que nossos objetos serão capazes de executar. Essas funções são projetadas para serem chamadas a partir de um objeto. Elas precisam de acesso ao estado do objeto - ou seja, aos valores de seus atributos. Por conta disso, o primeiro parâmetro de um método em Python sempre representará o objeto executando a ação.

Esse primeiro parâmetro pode receber qualquer nome, mas por padrão nós sempre utilizaremos a palavra self para representar o objeto executando a ação. No exemplo abaixo, o método dizer\_ola acessa o atributo nome do objeto que o executar. Falaremos mais sobre essa sintaxe do pontinho em breve.

class Jogador:

...

def dizer\_ola(self):

print(f'Olá, meu nome é {self.nome}.')

1.3. Método construtor

Você pode criar qualquer tipo de método que faça sentido em sua modelagem. Eles podem possuir os nomes de sua preferência, receber parâmetros diversos (além do self) e retornar ou não valores.

Mas existe um conjunto de métodos especiais conhecidos como métodos mágicos. Eles possuem nomes específicos e permitem adicionar funcionalidades específicas do Python aos objetos de sua classe. Você pode reconhecer um método mágico graças ao padrão dunder (double underscore): o nome de todos esses métodos padrão segue a sintaxe: \_\_nome\_\_. Abordaremos métodos mágicos diversos em outros capítulos, mas por hora estamos interessados em um método em particular: o método construtor.

O construtor é um método executado de maneira automática sempre que um novo objeto de uma classe é criado. Por conta disso, ele é utilizado para realizar a inicialização do objeto, ou seja, a criação de seus atributos e atribuição de um valor inicial. Criando atributos dentro do construtor temos a garantia de que todos os objetos da classe terão os mesmos atributos. Esse método deve ser implementado com o nome \_\_init\_\_:

class Jogador:

def \_\_init\_\_(self, nome):

self.nome = nome

self.pontuacao = 0

def dizer\_ola(self):

print(f'Olá, meu nome é {self.nome}.')

Note que podemos passar valores para atributos como parâmetros no construtor, mas isso não é obrigatório. A lógica do seu problema pode ditar como o valor de um atributo é determinado. No exemplo acima, nome deve ser fornecido, mas pontuacao sempre começa valendo zero - provavelmente porque não faria sentido uma partida iniciar com um jogador já possuindo pontos.

Vale sempre reforçar que cada objeto terá os seus próprios valores para os atributos. O \_\_init\_\_ irá garantir que todo objeto da classe Jogador tenha nome e pontuação, mas cada jogador terá o seu próprio nome e o seu próprio valor de pontuação.

2. Objetos

Tendo uma classe pronta podemos começar a instanciar (criar) objetos. Fazemos isso chamando o nome da classe como se fosse uma função, e entre parênteses passamos os atributos esperados pelo \_\_init\_\_, exceto pelo self. O self é passado de maneira automática e implícita. Sendo assim, na classe Jogador acima precisamos passar apenas o nome.

player1 = Jogador('Mario')

player2 = Jogador('Luigi')

As variáveis player1 e player2 no exemplo acima são dois objetos da classe Jogador.

2.1. Acessando atributos

Para acessar atributos utilizamos a sintaxe nome\_do\_objeto.nome\_do\_atributo. No exemplo abaixo, vamos modificar o atributo pontuacao de ambos os objetos e ver os atributos de ambos na tela:

player1.pontuacao = 5

player2.pontuacao = 10

print(f'Pontuação do {player1.nome}: {player1.pontuacao}')

print(f'Pontuação do {player2.nome}: {player2.pontuacao}')

2.2. Invocando métodos

Para invocar um método, vamos combinar a sintaxe de pontinho que vimos acima com a sintaxe de chamada de função: nome\_do\_objeto.nome\_do\_metodo(parametros).

Lembre-se que o primeiro parâmetro de um método é o self, e o self nunca é passado explicitamente: o Python vai pegar o objeto que veio antes do ponto e passar automaticamente como primeiro parâmetro. Você precisa passar a partir do segundo parâmetro do método, caso haja. No exemplo abaixo os nossos jogadores irão executar o método dizer\_ola:

player1.dizer\_ola()

player2.dizer\_ola()

Note que em cada execução o método usou as informações (no caso, o atributo nome) do jogador que o invocou.

Você já viu essa sintaxe antes:

lista.append(novo\_valor) maiuscula = nome.lower()

Em Python, praticamente tudo é um objeto e possui uma classe. Quando utilizamos a função type para visualizar o tipo de uma variável, a resposta sempre tem a palavra class, indicando a qual classe aquele objeto pertence.

Listas são objetos da classe list, e as funções append, remove, insert, pop etc são métodos dessa classe. O mesmo vale para a classe str e os métodos upper, lower, split, replace etc.

3. Mais um exemplo

Vejamos mais um exemplo para revisar todos os conceitos.

Criando uma classe:

import random

class Usuario:

# Método construtor - aqui criamos os nossos atributos

def \_\_init\_\_(self, nome, cpf, email):

# ao fazer self.atributo, estamos acessando um atributo do objeto sendo criado

self.nome = nome

self.cpf = cpf

self.login = email

# nem todo atributo precisa vir como parâmetro para o construtor

# o atributo abaixo vai receber um valor aleatorio:

self.senha = str(random.randint(100000, 999999))

def fazer\_login(self, login, senha):

# ao fazer self.atributo, estamos acessando um atributo do objeto executando a ação

if login == self.login and senha == self.senha:

print(self.nome, 'logado com sucesso!')

else:

print('Erro! Login ou senha incorretos!')

Agora que temos nossa classe definida, podemos criar objetos pertencentes a essa classe. Não iremos chamar o construtor explicitamente. Utilizaremos o nome da classe como se fosse uma função, e o seu retorno será o novo objeto:

professor1 = Usuario('Rafael', 12345678912, 'rafael@ada.com.br')

professor2 = Usuario('Romero', 98765432198, 'romero@ada.com.br')

Note que não passamos o self. Ele é sempre passado implicitamente.

Podemos acessar os atributos utilizando o ponto:

print(professor1.nome)

print(professor2.login)

Resultado na tela:

Rafael

romero@ada.com.br

Podemos também chamar métodos utilizando a sintaxe do ponto. Mas devemos sempre nos lembrar que:

Métodos são funções. Logo, eles possuem parênteses (e possivelmente parâmetros).

O self é sempre passado implicitamente e será o objeto à esquerda do pontinho.

professor1.fazer\_login('rafael@ada.com.br', '123456')