**1 - Criando a classe produto**

[001]**class Parametro**():

[002] **def \_\_init\_\_**(self, nome, minimo, maximo):

[003] self.nome = nome

[004] self.minimo = minimo

[005] self.maximo = maximo

[007]**if** Toggle:

[008] lista\_parametros = []

[009] lista\_parametros.**append**(**Parametro**('Largura', 0.00, 11.00))

[010] lista\_parametros.**append**(**Parametro**('Altura', 2.00, 22.00))

[011] **for** parametro **in** lista\_parametros:

[012] **print**(parametro.nome)

Há vários aspectos a serem observados aqui, mas não se preocupe. Você verá essa estrutura ao longo deste capítulo e terá bastante tempo para se acostumar com ela. Em u definimos uma classe chamada Dog. Por convenção, nomes com a primeira letra maiúscula referem-se a classes em Python. Os parênteses na definição da classe estão vazios porque estamos criando essa classe do zero. Em v escrevemos uma docstring que descreve o que essa classe faz.

**1.1 - Método \_\_init\_\_()**

Uma função que faz parte de uma classe é um método. Tudo que aprendemos sobre funções também se aplica aos métodos; a única diferença prática, por enquanto, é o modo como chamaremos os métodos. O método \_\_init\_\_() em w é um método especial que Python executa automaticamente sempre que criamos uma nova instância baseada na classe Dog. Esse método tem dois *underscores* no início e dois no final – uma convenção que ajuda a evitar que os nomes default de métodos Python entrem em conflito com nomes de métodos criados por você.

Definimos o método \_\_init\_\_() para que tenha três parâmetros: self, maximo e minimo. O parâmetro self é obrigatório na definição do método e deve estar antes dos demais parâmetros. Deve estar incluído na definição, pois, quando Python chama esse método \_\_init\_\_() depois (para criar uma instância de Dog), a chamada do método passará o argumento self automaticamente. Toda chamada de método associada a uma classe passa self, que é uma referência à própria instância, de modo automático; ele dá acesso aos atributos e métodos da classe à instância individual. Quando criamos uma instância de Dog, Python chamará o método \_\_init\_\_() da classe Dog. Passaremos um nome e uma idade como argumentos para Dog(); self é passado automaticamente, portanto não é preciso especificá-lo. Sempre que quisermos criar uma instância da classe Dog forneceremos valores apenas para os dois últimos parâmetros, que são name e age.

As duas variáveis definidas em x têm o prefixo self. Qualquer variável prefixada com self está disponível a todos os métodos da classe; além disso, podemos acessar essas variáveis por meio de qualquer instância criada a partir da classe. self.name = name usa o valor armazenado no parâmetro name e o armazena na variável name, que é então associada à instância criada. O mesmo processo ocorre com self.age = age. Variáveis como essas, acessíveis por meio de instâncias, são chamadas de

atributos.

A classe Dog tem dois outros dois métodos definidos: sit() e roll\_over() y. Como esses métodos não precisam de informações adicionais como um nome ou uma idade, simplesmente os definimos com um parâmetro self. As instâncias que criarmos posteriormente terão acesso a esses métodos. Em outras palavras, elas terão a capacidade de sentar e rolar.

Por enquanto, sit() e roll\_over() não fazem muito. Apenas exibem uma mensagem dizendo que o cachorro está sentando ou rolando. No entanto, o conceito pode ser estendido para situações realistas: se essa classe fizesse parte de um jogo de computador de verdade, esses métodos conteriam código para fazer a animação de um cachorro sentando e rolando. Se essa classe tivesse sido escrita para controlar um robô, esses métodos direcionariam os movimentos para fazer um cachorro-robô sentar e rolar.

Criando uma instância a partir de uma classe

Pense em uma classe como um conjunto de instruções para criar uma instância. A classe Dog é um conjunto de instruções que diz a Python como criar instâncias individuais que representem cachorros específicos.

Vamos criar uma instância que represente um cachorro específico:

class Dog(): --trecho omitido--

u my\_dog = Dog('willie', 6)

v print("My dog's name is " + my\_dog.name.title() + ".") w print("My dog is " + str(my\_dog.age) + " years old.") A classe Dog que usamos aqui é aquela que acabamos de escrever no exemplo anterior. Em u dizemos a Python para criar um cachorro de nome 'willie' e idade igual a 6. Quando Python lê essa linha, ele chama o método \_\_init\_\_() de Dog com os argumentos 'willie' e 6. O método \_\_init\_\_() cria uma instância que representa esse cachorro em particular e define os atributos name e age com os valores que fornecemos.

Esse método não tem uma instrução return explícita, mas Python devolve automaticamente uma instância que representa esse cachorro. Armazenamos

essa instância na variável my\_dog. A convenção de nomenclatura é útil nesse

caso: em geral, podemos supor que um nome com a primeira letra maiúscula

como Dog refere-se a uma classe, enquanto um nome com letras minúsculas

como my\_dog refere-se a uma única instância criada a partir de uma classe.

Acessando atributos

Para acessar os atributos de uma instância utilize a notação de ponto. Em v acessamos o valor do atributo name de my\_dog escrevendo:

my\_dog.name

A notação de ponto é usada com frequência em Python. Essa sintaxe mostra como Python encontra o valor de um atributo. Nesse caso, o interpretador olha para a instância my\_dog e encontra o atributo name associado a ela. É o mesmo atributo referenciado como self.name na classe Dog. Em w usamos a mesma abordagem para trabalhar com o atributo age. Em nossa primeira instrução print, my\_dog.name.title() faz com que 'willie' – o valor do atributo name de my\_dog – comece com uma letra maiúscula. Na segunda instrução print, str(my\_dog.age) converte 6 – o valor do atributo age de my\_dog – em uma string.

A saída é um resumo do que sabemos sobre my\_dog: My dog's name is Willie.

My dog is 6 years old.

[pág. 203 – curso intensive]