INTELIGÊNCIA ARTIFICIAL

Fabricio Machado da Silva

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Introdução ao Python II

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer a criação de funções no Python.
- Definir classes no Python.
- Desenvolver programas orientados aos objetos no Python.

Introdução

As funções são blocos de código e realizam tarefas que geralmente precisam ser executadas diversas vezes em uma aplicação. Quando surge essa necessidade, para que várias instruções não sejam repetidas, elas se agrupam em uma função, à qual é dada um nome e pode ser chamada ou executada em diferentes partes do programa. O Python é uma linguagem de programação orientada a objetos (POO), com recursos que dão suporte a esta, como a criação e utilização de classes.

Neste capítulo, você estudará as funções e classes na linguagem Python, sua definição e seu uso na prática, bem como o desenvolvimento de POO.

Funções

As funções são definidas como sub-rotinas que executam um objetivo em particular. Todas as linguagens de programação utilizadas atualmente possuem formas de criá-las, como Python, C++, Java, C# e *Object Pascal* (*Delphi*). Além de o programador poder criar as próprias funções, a linguagem de programação tem funções que já estão incluídas na biblioteca padrão com o objetivo facilitar o trabalho dele. Seus exemplos são as funções matemáticas para cálculo de seno, cosseno e tangente, bem como as funções de manipulação de texto (*substring*), que permitem manipular caracteres (BARRY; GRIFFITHS, 2010).

O Python disponibiliza diversas funções internas (*built-in*), como as mencionadas anteriormente, porém, você pode criar outras. Um dos principais propósitos das funções é agrupar os códigos que serão executados várias vezes, sendo que, sem uma função definida, seria necessário copiá-los e colá-los cada vez que fossem usados, resultando em um programa de aspecto mais poluído e uma manutenção bem mais complexa. Para entender melhor essa questão, analise o código da seguinte função (adaptado de SWEIGART, 2015, p. 95):

```
def ola():
    print('Olá!')
    print('Olá!!!')
    print('Alguém aí?')

ola()
ola()
ola()
```

As linhas contendo ola() após a função são as chamadas de função. No código, uma chamada de função é composta simplesmente do seu nome seguido de parênteses, possivelmente com alguns argumentos entre eles. Quando alcança essas chamadas, a execução do programa segue para a linha inicial da função e começa a executar o código a partir daí. Já ao alcançar o final da função, ela retorna à linha em que essa função foi chamada e continua percorrendo o código como anteriormente.

Caso não se defina uma função, para a execução desse código três vezes, seria necessário copiá-lo e colá-lo cada vez que ele fosse usado, assim, o programa teria o seguinte aspecto (adaptado de SWEIGART, 2015, p. 95–96):

```
print('Olá!')
print('Olá!!!')
print('Alguém aí?')
print('Olá!')
print('Olá!!!')
print('Alguém aí?')
print('Olá!')
print('Olá!!!')
print('Alguém aí?')
```

Na Figura 1, você pode observar a função para exibir o famoso *hello world*.

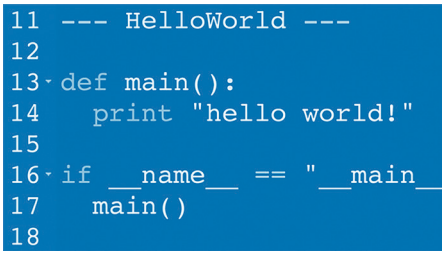

Figura 1. Função para exibir o famoso *hello world*.
Fonte: Spak (2016, documento *on-line*).

Instruções *def* para passagem de parâmetros

Ao chamar as funções da biblioteca padrão, como print() ou len(), passa-se valores (ou argumentos, nesse contexto) ao digitá-los entre os parênteses. Você ainda pode definir outras funções que aceitem argumentos. Como exemplo, avalie o seguinte código (adaptado de SWEIGART, 2015, p. 96):

```
def ola(name):
    print('Olá ' + name)

ola('Alice')
ola('Carlos')
```

A definição da função ola() nesse programa contém um parâmetro chamado name, sendo que parâmetro é uma variável em que se armazena um argumento ao chamar uma função. Na primeira vez que se chama a função ola(), isso ocorre com o argumento 'Alice'. A execução do programa entra na função, e a variável name é automaticamente definida com 'Alice', exibido pela instrução print(). Um aspecto importante sobre um parâmetro é que se esquece o valor armazenado nele quando a função retorna. Por exemplo, ao adicionar print(name) depois de ola('Carlos') no programa anterior, um NameError será gerado pelo programa, pois não há uma variável name, a qual foi destruída

após o retorno da chamada de função ola('Carlos'), portanto, print(name) faria referência à variável name inexistente.

Valor de retorno da função (*return*)

Quando se chama a função len() e lhe passa um argumento como 'Olá', a chamada de função será avaliada como o valor inteiro três, que é o tamanho da *string* passada. Em geral, o valor com o qual uma chamada é avaliada se denomina valor de retorno da função. Assim, ao criar uma função com a instrução *def*, pode-se especificar qual deve ser o valor de retorno com uma instrução *return*, que se constitui de:

- Palavra-chave *return*.
- Valor ou expressão que a função deve retornar.

Quando se usa uma expressão com a instrução *return*, ela é avaliada com o valor de retorno. Por exemplo, o programa a seguir define uma função que retorna uma *string* diferente de acordo com o número passado a ela como argumento (adaptado de SWEIGART, 2015, p. 97–98):

```
import random
def getAnswer(answerNumber):
    if answerNumber == 1:
            return 'It is certain'
       elif answerNumber == 2:
            return 'It is decidedly so'
       elif answerNumber == 3:
            return 'Yes'
       elif answerNumber == 4:
            return 'Reply hazy try again'
       elif answerNumber == 5:
            return 'Ask again later'
       elif answerNumber == 6:
            return 'Concentrate and ask again'
       elif answerNumber == 7:
            return 'My reply is no'
       elif answerNumber == 8:
```

```
            return 'Outlook not so good'
        elif answerNumber == 9:
            return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

Em Python, há o *None*, que representa a ausência de um valor e é o único valor do tipo de dado *NoneType* — outras linguagens de programação também o chamam de *null*, *nil* ou *undefined*. Assim como os valores booleanos *True* e *False*, ele deve ser digitado com uma letra N maiúscula. Esse valor pode ser útil quando houver a necessidade de armazenar algo que não deva ser confundido com um valor real em uma variável. Um local em que se usa None é como valor de retorno de print(), cuja função exibe um texto na tela e não precisa retornar nada, como len() ou input() o fazem. No entanto, como todas as chamadas de função devem ser avaliadas com um valor de retorno, print() retorna *None*.

Escopos local e global

As variáveis e os parâmetros atribuídos em uma função chamada existem no seu escopo local, já as variáveis que recebem o valor fora de todas as funções ocorrem no escopo global. Uma variável que esteja no escopo local é chamada de variável local, e aquela que existe no escopo global se denomina variável global. Elas devem ser de um ou outro tipo e não podem ser, ao mesmo tempo, local e global (BARRY; GRIFFITHS, 2010).

Pense no escopo como um contêiner para variáveis que, ao ser destruído, todos os valores armazenados nelas são esquecidos. Há apenas um escopo global, criado quando seu programa se inicia e que, ao terminar, é destruído e esquece todas as suas variáveis. Se não fosse dessa forma, na próxima vez que você executasse o programa, as variáveis se lembrariam dos valores da última execução. Já um escopo local é criado sempre que se chama uma função, no qual existirá qualquer variável que receber um valor nessa função. Quando a função retornar, esse escopo será destruído, e as variáveis esquecidas. Na próxima vez que ela for chamada, as variáveis locais não se lembrarão dos valores armazenados na última vez que se usou essa função.

Fique atento

Em geral, sempre evite duplicar códigos, pois, se algum dia você decidir atualizá-lo, por exemplo, e encontrar um *bug* que deva ser corrigido, será necessário alterar seu código em todos os locais em que o copiou.

Classes

Uma classe representa uma categoria de objeto, por exemplo, a classe carro envolve todos os tipos de carro, mas cada carro ou objeto implementa seus atributos e métodos. Objetos são abstrações computacionais que representam entidades, com as qualidades (atributos) e as ações (métodos) que elas podem realizar. A classe é ainda a estrutura básica do paradigma de orientação aos objetos, que se refere ao seu tipo, um modelo a partir do qual esses objetos serão criados. Na Figura 2, você pode observar uma estrutura de classes.

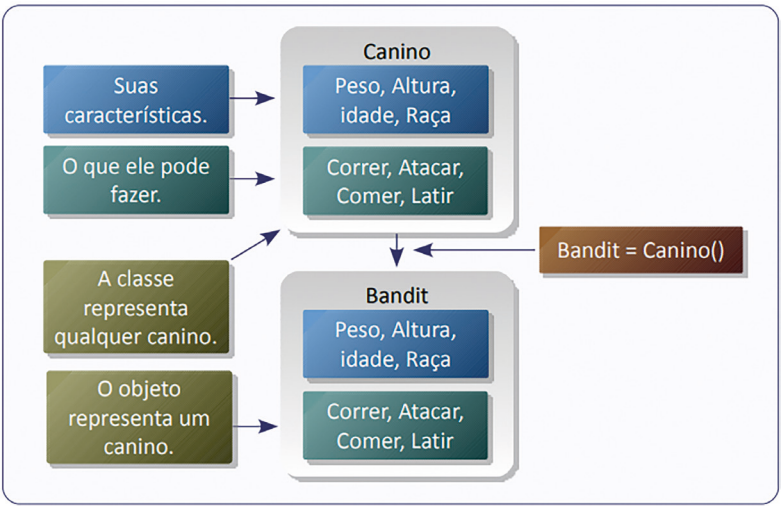

Figura 2. Exemplo de uma estrutura de classes.

Fonte: Borges (2014, p. 109).

Como você viu no exemplo da Figura 2, a classe canino descreve as características e ações dos caninos em geral, já o objeto Bandit representa um canino específico. Os atributos envolvem as estruturas de dados que armazenam informações sobre o objeto; os métodos, por sua vez, são funções associadas ao objeto, que descrevem como ele se comporta.

No Python, os objetos são criados a partir das classes por meio de atribuição, sendo uma instância delas, que possui características próprias. Ao criar um deles, executa-se o construtor da classe, que é um método especial, denominado __new__(). Após a chamada do construtor, o método __init__() também é chamado para inicializar uma nova instância.

Algumas características da POO na linguagem Python são:

- Quase tudo é objeto, mesmo os tipos básicos, como números inteiros.
- Tipos e classes são unificados.
- Operadores são, na verdade, chamadas para métodos especiais.
- Classes são abertas, menos para os tipos *builtins*.

Os métodos especiais são identificados por nomes no padrão __metodo__() (dois sublinhados no início e no final do nome) e definem como os objetos derivados da classe se comportarão em situações específicas, como na sobrecarga de operadores.

Classes no Python

Como já foi citado, as classes são o principal recurso da POO, que tem como objetivo criar modelos que representem objetos da vida real, com atributos e métodos que atendam a essa representação (LUTZ; ASCHER, 2007). A seguir, você pode conferir um exemplo na prática que utiliza a linguagem Python e cria uma classe para representar animais de estimação, a qual teria os seguintes atributos:

- Nome.
- Espécie.
- Nome do dono.

Observe a seguir o código dessa classe, que se chama AnimalEstimacao:

```
class AnimalEstimacao():

    def __init__(self, nome, especie, dono):
```

```
        self.nome = nome
        self.especie = especie
        self.dono = dono
```

O *self* que aparece na frente de cada atributo e no primeiro parâmetro do método __init__() se refere ao objeto criado, assim, self.especie define um atributo da classe que é a espécie do objeto animal de estimação. Note que especie e self.especie são distintos, sendo uma variável e um atributo da classe respectivamente. No código a seguir, usa-se a classe AnimalEstimacao criando dois animais de estimação peludos e fofinhos, uma notação para acessar os atributos de cada classe.

```
import animal_estimacao as animal

peludo = animal.AnimalEstimacao('Peludo', 'cão', 'Alice')
print('Meu nome é:', peludo.nome, 'eu sou um', peludo.especie,
'e meu dono é:', peludo.dono)
print()

fofinho = animal.AnimalEstimacao('Fofinho', 'gato', 'Luís')
print('Meu nome é:', fofinho.nome, 'eu sou um', fofinho.especie,
'e meu dono é:', fofinho.dono)
```

Até aqui apenas se definiu atributos para a classe, agora serão desenvolvidas algumas ações e, para isso, se utiliza a definição de métodos para a classe AnimalEstimacao, como correr, brincar e comer.

```
class AnimalEstimacao():
def__init__ (self, nome, especie, dono):
        self.nome = nome
        self.especie = especie
        self.dono = dono

    def correr(self):
        print('{0} está correndo'.format(self.nome))

    def brincar(self):
        print('{0} está brincando'.format(self.nome))
```

```
    def comer(self):
        print('{0} está comendo'.format(self.nome))
```

Se você já sabe o que são as funções, fica mais simples de entender. As classes implementam atributos e métodos, sendo que estes executam ou implementam uma funcionalidade, assim como as funções. Portanto, a classe é um elemento que carrega diversas funções dentro de si.

Vamos a um exemplo utilizando classes (CAELUM ENSINO E INOVAÇÃO, 2019, p. 139):

```
class Ponto:

def__init__ (self, x, y):
        self.x = x
        self.y = y

    def__str__ (self):
return "({}, {})".format(self.x, self.y)

    def _ repr _ (self):
        return "Ponto({}, {})".format(self.x + 1, self.y + 1)

if _ name _ == ' _ main _ ':
    p1 = Ponto(1, 2)
    p2 = eval(repr(p1))

    print(p1)
    print(p2)
```

Se executarmos o código acima, temos:

```
(1, 2)
(2, 3)
```

Para concluir, deve-se entender que tanto __str__() como __repr__() retornam uma *string* que representa o objeto, mas com propósitos diferentes. O método __str__() é utilizado para apresentar mensagens para os usuários da classe, de forma mais amigável; já o __repr__() se usa para representar

o objeto de modo técnico, inclusive como comando válido do Python, no exemplo da classe Ponto.

Portanto, pode-se criar múltiplos objetos da mesma classe, os quais serão únicos. Eles têm o mesmo tipo, mas podem armazenar diferentes valores para suas propriedades individuais.

Saiba mais

Em Python, não existem variáveis e métodos privados, que apenas podem ser acessados a partir do próprio objeto. Em vez disso, usa-se uma convenção, um nome que comece com sublinhado (_) deve ser considerado parte da implementação interna do objeto e sujeito às mudanças sem aviso prévio. Além disso, essa linguagem oferece uma funcionalidade chamada *name mangling*, que acrescenta na frente dos nomes que iniciam com dois sublinhados (__), um sublinhado e o nome da classe.

Link

Saiba mais sobre a linguagem Python e como criar uma classe nela nos *links* a seguir.

https://qrgo.page.link/gfQFV

https://qrgo.page.link/CUwPK

Referências

BARRY, P.; GRIFFITHS, D. *Use a cabeça! Programação*: o guia do aprendiz de qualquer curso de programação. Rio de Janeiro: Alta Books, 2010. 440 p.

BORGES, L. E. *Python para desenvolvedores*. São Paulo: Novatec, 2014. 320 p.

CAELUM ENSINO E INOVAÇÃO. Herança e polimorfismo. *In*: CAELUM ENSINO E INOVAÇÃO. *Python e orientação a objetos*: curso PY-14. São Paulo: Caelum, 2019. p. 132–152. Disponível em: https://www.caelum.com.br/apostila-python-orientacao-objetos/heranca-e-classes-abstratas/. Acesso em: 1 jun. 2019.

LUTZ, M.; ASCHER, D. *Aprendendo Python*. 2. ed. Porto Alegre: Bookman; O'Reilly, 2007. 566 p.

SPAK, F. Funções em Python. *DevMedia*, Rio de Janeiro, 2016. Disponível em: https://www.devmedia.com.br/funcoes-em-python/37340. Acesso em: 1 jun. 2019.

SWEIGART, A. Funções. *In*: SWEIGART, A. *Automatize tarefas maçantes com Python*: programação prática para verdadeiros iniciantes. São Paulo: Novatec; No Starch Press, 2015. 604 p.

Leituras recomendadas

BRUECK, D.; TANNER, S. *Python 2.1 bible*: includes a complete Python language reference. New York: Hungry Minds, 2001. 731 p.

MATTHES, E. *Curso intensivo de Python*: uma introdução prática e baseada em projetos à programação. São Paulo: Novatec, 2016. 656 p.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS