Introdução à Orientação a Objetos
Parte 02

Prof. Fábio Procópio

Prof. João Nascimento

Técnico Integrado em Informática – Programação Estruturada e Orientada a Objetos

Relembrando...

- Na aula passada, iniciamos nossos estudos sobre Orientação a Objetos;
- Falamos sobre os conceitos de **classe**, **objeto**, **atributos** e **métodos**;
- Nesta aula, implementaremos os conceitos da aula passada.

Relembrando...

- Implementaremos alguns exemplos usando a classe Cachorro, definida na aula passada. Vamos relembrar o modelo da classe:

Cachorro
- nome - idade - raca
+ latir() + brincar()

Declaração de classes

- Em Python, a criação de classes dá-se por meio da palavra-chave **`class`**;
- Observe o exemplo da definição de uma classe chamada Cachorro;
- Para iniciarmos nossas implementações, crie um arquivo chamado **Cachorro.py** (por enquanto, ela nada faz):

```
class Cachorro:
    pass
```

Usado para dizer ao Python "passe esse bloco", uma espécie de "aqui, faça nada!"

Palavra-chave self

- Na definição de métodos, **`self`** é o primeiro parâmetro definido e **não deve ser esquecida**;

- `self` é quem viabiliza o acesso aos outros métodos de uma classe.

Declaração de métodos

- Para declarar um método, utilizamos a palavra-chave **`def`** (a mesma utilizada para criar funções ☺);

- Vamos declarar os métodos da classe Cachorro. Observe que, agora, a classe começa a ter funcionalidades:

```
class Cachorro:

    def latir(self):
        return "au au au"

    def brincar(self):
        return "Estou cavando o jardim."
```

Método construtor __init__()

ATENÇÃO: São **dois** *underlines*, antes e depois da palavra **init**.

- É chamado de método construtor e é utilizado para **inicializar o estado de um objeto**;
- Invocado quando cada nova instância de uma classe (objeto) é criada;
- Vamos criar o método construtor **`__init__()`** para a classe Cachorro (Cachorro.py):

```
class Cachorro:

    # ATENÇÃO: Antes e depois do método init, temos 2 underlines, isto é, __
    def __init__(self):
        pass

    def latir(self):
        return "au au au"

    def brincar(self):
        return "Estou cavando o seu jardim."
```

Instanciação de objetos

- Vamos criar um arquivo chamado **TestaClasses.py** o qual será utilizado para validar nossas classes:

```
from Cachorro import *

# Instanciando um objeto chamado dog
dog = Cachorro()

# Executando os métodos da classe Cachorro
print(dog.latir())
print(dog.brincar())
```

- Para que não ocorram erros de acesso, **salve** TestaClasses.py **na mesma pasta** do arquivo Cachorro.py;

- Não devemos esquecer de fazer a **importação da classe Cachorro** para que ela possa ser usada por TestaClasses.py

Declaração de atributos

- Vamos usar a classe Cachorro para exemplificar. Sabemos que existem 3 atributos: nome, idade e raca;

- Modificaremos o método construtor de modo que os atributos de Cachorro sejam enviados como argumento para **`__init__()`**:

```
class Cachorro:

   def __init__(self, p_nome, p_idade, p_raca):
      self.nome  = p_nome
      self.idade = p_idade
      self.raca  = p_raca

   def latir(self):
      return "au au au"

   def brincar(self):
      return "Estou cavando o seu jardim."
```

Exercício Resolvido 01 – 1 de 2

- Modifique a classe Cachorro adicionando o método **`imprimir()`**:

```
class Cachorro:

   def __init__(self, p_nome, p_idade, p_raca):
      self.nome  = p_nome
      self.idade = p_idade
      self.raca  = p_raca

   def latir(self):
      return "au au au"

   def brincar(self):
      return "Estou cavando o seu jardim."

   def imprimir(self):
      dados  = "Nome.: {}\n".format(self.nome)
      dados += "Idade: {} anos\n".format(self.idade)
      dados += "Raça.: {}".format(self.raca)
      return dados
```

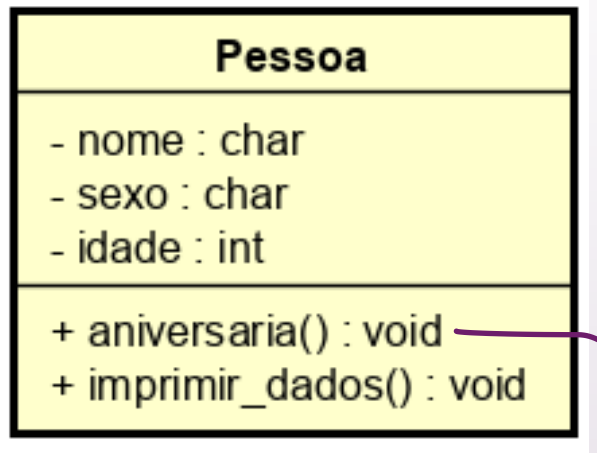
Exercício Resolvido 01 – 2 de 2

- Modifique TestaClasses.py incluindo o seguinte trecho de código:

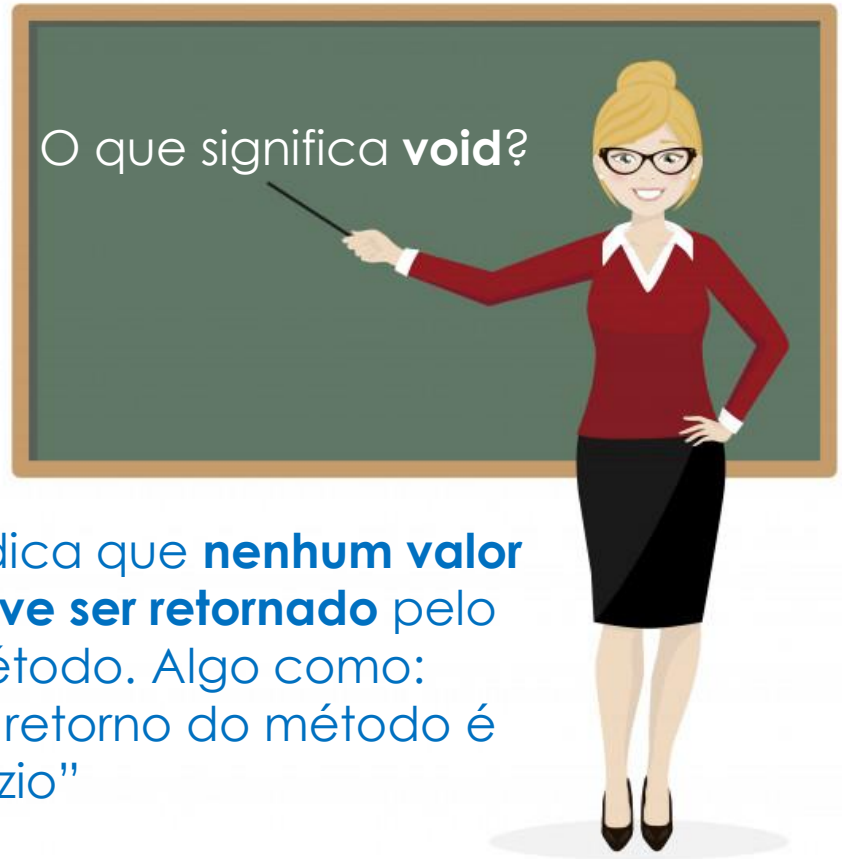
```
from Cachorro import *

'''Observe que informamos 3 argumentos para instanciar
um objeto de Cachorro. Isso foi feito porque o seu método
construtor aguarda a passagem desses parâmetros'''

dog = Cachorro("core", 11, "SRD")

print("********** Dados do cachorro **********")
print(dog.imprimir())
print("***************************************")
```

Exercício Resolvido 02 – 1 de 3

- Implemente a seguinte classe:



aniversaria() deve fazer com que o atributo **idade** seja incrementado cada vez que ele for executado.



Indica que **nenhum valor deve ser retornado** pelo método. Algo como: "o retorno do método é vazio"

Exercício Resolvido 02 – 2 de 3

- Crie uma classe chamada Pessoa e salve-a no arquivo **Pessoa.py**:

```
class Pessoa:
    def __init__(self, p_nome, p_sexo, p_idade):
      self.nome  = p_nome
      self.sexo  = p_sexo
      self.idade = p_idade

    def imprimir_dados(self):
      dados  = "********** DADOS **********\n"
      dados += "Nome.: {}\n".format(self.nome)
      dados += "Sexo.: {}\n".format(self.sexo)
      dados += "Idade: {} anos\n".format(self.idade)
      dados += "**************************"
      print(dados)

    def aniversaria(self):
      # Toda vez que a pessoa aniversaria, sua idade é acrescida em 1 ano
      self.idade += 1
      print("Parabéns, {} pelos seus {} anos.".format(self.nome, self.idade))
```

Exercício Resolvido 02 – 2 de 3

- Em TestaClasses.py, escreva o seguinte código:

```
from Pessoa import *

paolla = Pessoa("Paolla de Oliveira", "F", 37)
paolla.aniversaria()
paolla.aniversaria()
paolla.aniversaria()
paolla.imprimir_dados()
```

Exercício Resolvido 03 – 1 de 4

- Implemente a classe Porta.

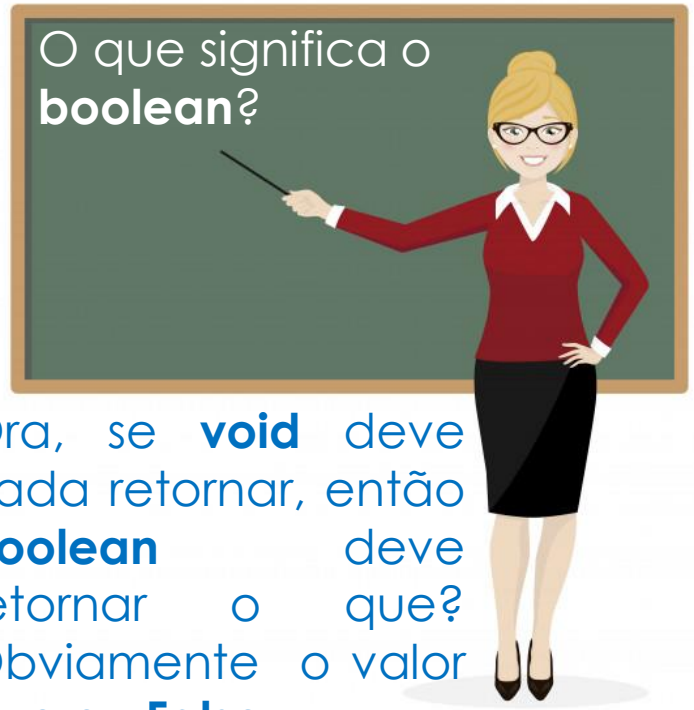
Porta
- aberta : boolean - cor : char - dimensaoX : float - dimensaoY : float - dimensaoZ : float
+ pinta(p_cor : char) : void + mostra_estado() : boolean + abre() : void + fecha() : void

Método **`mostra_estado()`**
Verifica se a porta está aberta ou fechada

Método **`abre()`**
Só poderá abrir a porta se ela estiver fechada.
Caso contrário, um erro deve ser exibido

Método **`fecha()`**
Só poderá fechar a porta se ela estiver aberta.
Caso contrário, um erro deve ser exibido

ATENÇÃO
Antes de abrir ou fechar a porta, use o método **`mostra_estado()`**



Ora, se **void** deve nada retornar, então **boolean** deve retornar o que? Obviamente o valor **True** ou **False**.

Exercício Resolvido 03 – 2 de 4

- Crie uma classe chamada Porta e salve-a no arquivo **Porta.py**:

```
class Porta:
    def __init__(self, p_dimensaoX, p_dimensaoY, p_dimensaoZ):
      self.aberta    = False # Sempre que criada uma porta ela estará fechada
      self.cor       = None  # Sempre que criada uma porta ela estará sem cor
      self.dimensaoX = p_dimensaoX
      self.dimensaoY = p_dimensaoY
      self.dimensaoZ = p_dimensaoZ

    def pinta(self, p_cor):
      self.cor = p_cor
      print("Porta pintada de {}.".format(self.cor))

    def mostra_estado(self):
      return self.aberta
```

Exercício Resolvido 03 – 3 de 4

- Continuação do arquivo **Porta.py**:

```
# Continuação...
def abre(self):
  if self.mostra_estado() == False:
     self.aberta = True
     print("OK: Porta aberta.")
  else:
     print("ERRO: Porta já está aberta.")

def fecha(self):
  if self.mostra_estado() == True:
     self.aberta = False
     print("OK: Porta fechada.")
  else:
     print("ERRO: Porta já está fechada.")
```

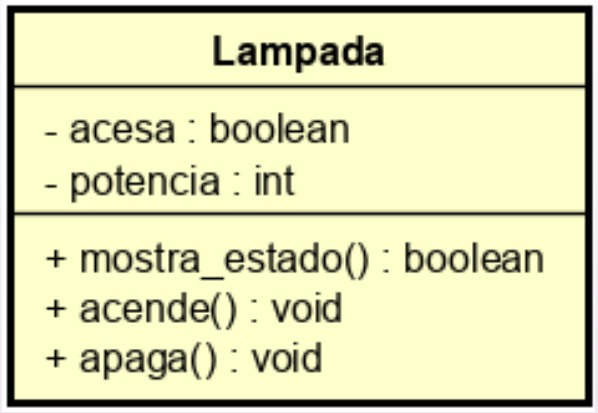
Exercício Resolvido 03 – 4 de 4

- Crie um arquivo chamado **TestaPorta.py**:

```
from Porta import *

p1 = Porta(0.9, 2.5, 0.05)
p1.pinta("cinza")
p1.abre()
p1.fecha()

# Observe que será gerado um erro porque a porta já foi fechada
p1.fecha()
```

Exercício de Fixação 01

- Implemente a classe Lampada.



Método **`mostra_estado()`**
Verifica se a lâmpada está acesa ou apagada e retorna um valor boolean

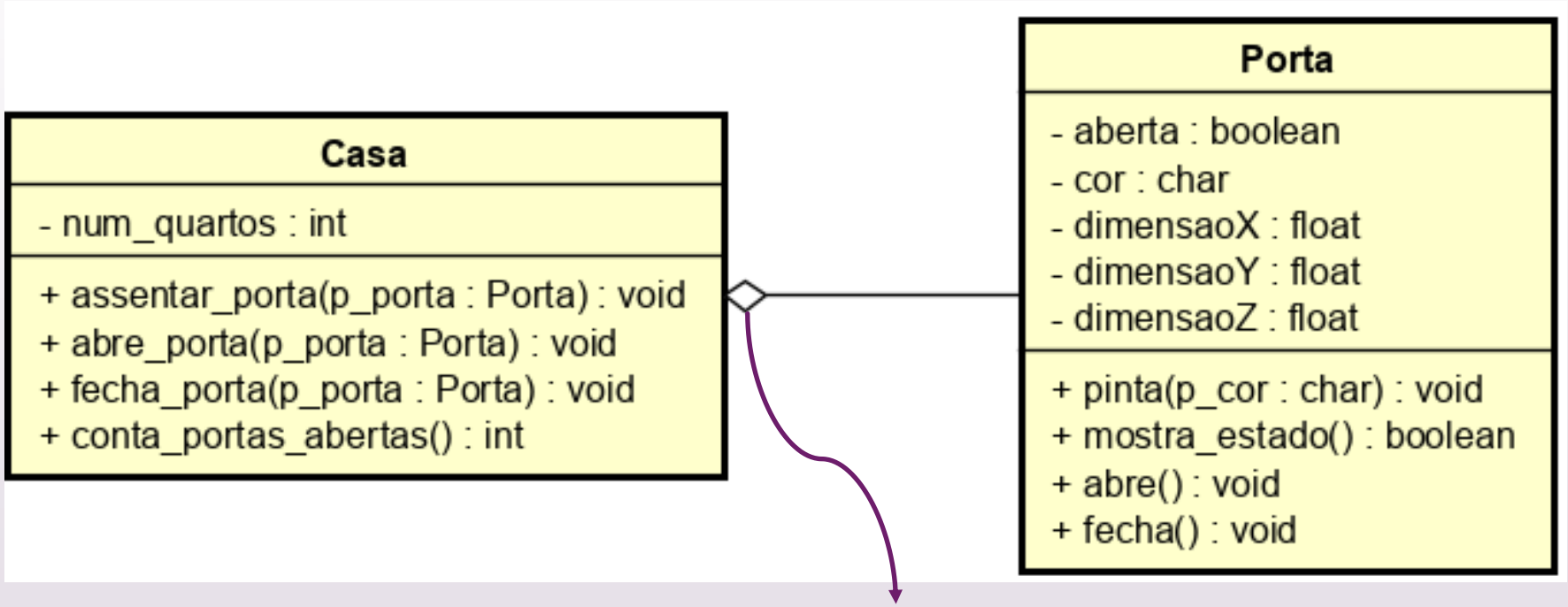
Método **`acende()`**
Só poderá acender a lâmpada se ela estiver apagada. Caso contrário, um erro deve ser disparado

Método **`apaga()`**
Só poderá apagar a lâmpada se ela estiver acesa. Caso contrário, um erro deve ser disparado

ATENÇÃO
Antes de acender ou apagar a lâmpada, use o método **`mostra_estado()`**

Exercício Resolvido 05 – 1 de 4

- Implemente o diagrama de classe abaixo considerando a associação do tipo Agregação entre as classes Casa e Porta.



Esse losango "aberto" significa **Agregação**, a qual tenta demonstrar que as informações de um objeto-todo precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (objeto-parte).

Exercício Resolvido 05 – 2 de 4

- Crie uma classe chamada Casa e salve-a no arquivo **Casa.py**:

```
from Porta import *

class Casa:
    def __init__(self, p_num_quartos):
        self.num_quartos = p_num_quartos
        self.portas      = []

    def assentar_porta(self, p_porta):
        self.portas.append(p_porta)

    def abre_porta(self, p_porta):
        for indice, obj_porta in enumerate(self.portas):
            if obj_porta == p_porta:
                self.portas[indice].abre()
```

Exercício Resolvido 05 – 3 de 4

- Continuação do arquivo **Casa.py**:

```
# Continuação da classe Casa

   def fecha_porta(self, p_porta):
       for indice, obj_porta in enumerate(self.portas):
           if obj_porta == p_porta:
               self.portas[indice].fecha()

   def conta_portas_abertas(self):
       abertas = 0
       for obj_porta in self.portas:
           if obj_porta.mostra_estado() == True:
              abertas += 1
       return abertas
```

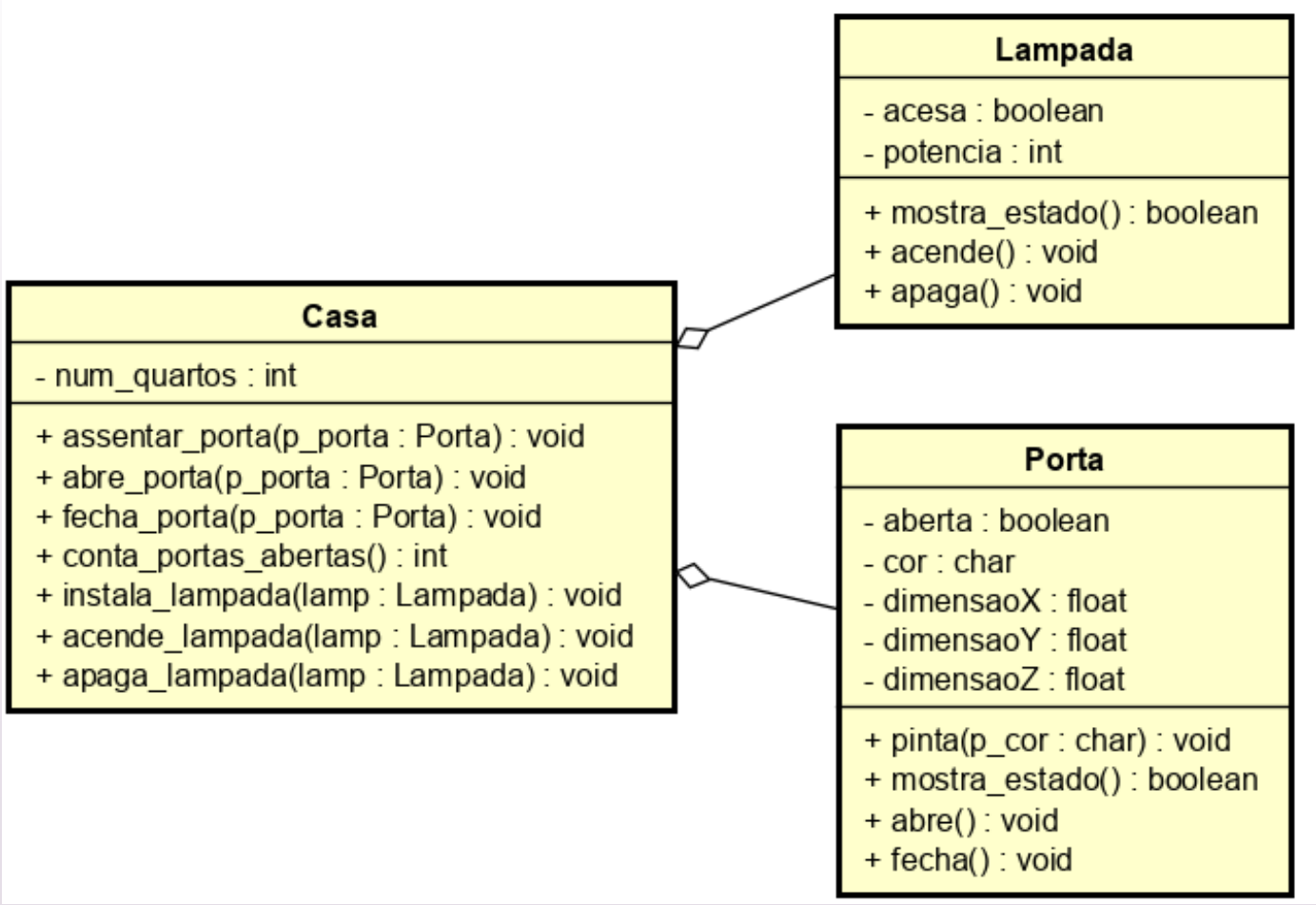
Exercício Resolvido 05 – 4 de 4

- Crie um arquivo chamado **TestaCasa.py**:

```
from Casa import *

p1 = Porta(0.9, 2.5, 0.05)
p2 = Porta(0.9, 2.5, 0.05)
p3 = Porta(0.9, 2.5, 0.05)

c  = Casa(3) #Casa possui 3 quartos

c.assentar_porta(p1)
c.assentar_porta(p2)
c.assentar_porta(p3)

c.abre_porta(p1)
c.abre_porta(p2)
print("Portas abertas: {}".format(c.conta_portas_abertas()))

# Será gerado um erro porque a porta p1 já está aberta
c.abre_porta(p1)
```

Exercício de Fixação 02

- Modifique a classe Casa, observando a nova associação com a classe **`Lampada`** e os métodos **`instala_lampada()`**, **`acende_lampada()`** e **`apaga_lampada()`**.



Canal no Youtube

Em nosso canal no Youtube, você pode acessar uma playlist que preparamos com várias vídeo-aulas falando sobre o assunto. Veja:

- Relacionamento de Agregação

Referências

1. DEVMEDIA. Introdução à Programação Orientada a Objetos em Java. Disponível em: https://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452. Acessado em: 12 jun. 2019.

2. Silva, Régis. Introdução a Classes e Métodos em Python (básico). Acessado em: 20 Set. 2016. Disponível em: http://pythonclub.com.br/introducao-classes-metodos-python-basico.html. Acessado em: 12 jun. 2019.

3. PythonBrasil, Python e Programação Orientada a Objetos. Disponível em: http://wiki.python.org.br/ProgramacaoOrientadaObjetoPython. Acessado em: 12 jun. 2019.