Orientação a Objetos na Linguagem Python

1. Definindo uma classe

O nome da classe precisa ser um substantivo e a primeira letra deverá estar em Maiúsculo.

```
class Cachorro:
    pass
```

2. Definindo o construtor

O construtor é a primeira função invocada quando se cria um objeto de uma classe.

```
class Cachorro:
    def __init__(self, nome, raca, idade):
        self.nome = nome
        self.raca = raca
        self.idade = idade
```

Dizemos que *nome*, *raça* e *idade* são os **atributos** da classe *Cachorro*

3. Criando objetos

Um objeto é uma **instância** de uma classe

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog2 = Cachorro("Lili", "poodle", 2)
```

4. Acessando dados dos objetos

```
print(dog1.nome)
print(dog2.idade)
```

Devemos evitar fazer isso!

5. Modificando os valores dos atributos de um objeto

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog1.nome = "Bob"
```

Muito arriscado. Evitar esse procedimento!

6. Definindo métodos

Um método é uma função dentro da classe. Sempre devemos escrever o nome do método com a primeira letra em minúscula.

```
class Cachorro:
    def __init__(self, nome, raca, idade):
        self.nome = nome
        self.raca = raca
        self.idade = idade

    def falar(self):
        print("au au!")

    def getIdade(self):
        return self.idade
```

7. Acessando os métodos

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog1.falar()
dog1.getIdade()
```

8. Alterando a visibilidade dos atributos de uma classe

Para evitar que os atributos de uma classe possam ser acessados e/ou alterados fora da classe (igual fizemos nos tópicos 4 e 5), devemos alterar a **visibilidade** dos atributos. Para isso, utilizamos dois *underscores* ('__') antes do nome do atributo. Exemplo:

```
class Cachorro:
    def __init__(self, nome, raca, idade):
        self.__nome = nome
        self.raca = raca
        self.idade = idade
```

No exemplo acima, o atributo *nome* foi definido como **privado**. Isso significa que não conseguiremos **acessar** o seu valor fora da classe. Veja o exemplo a seguir:

```
dog1 = Cachorro("Rex", "bulldog", 3)
print(dog1.__nome)
```

Ao executar o código acima, o interpretador irá acusar que o atributo *nome* **não existe** na classe *Cachorro*. Mas isso não garante que ninguém possa acessá-lo. No Python não existem atributos realmente privados, ele apenas alerta que você não deveria tentar acessar este atributo, ou modificá-lo. Para acessá-lo, podemos fazemos:

```
print(dog1._Cachorro__nome)
```

Devemos evitar fazer isso!

9. Métodos Setters e Getters

Ao invés de chamar os atributos de uma classe diretamente (igual fizemos nos tópicos 4 e 5), devemos implementar os métodos *setters* e *getters* que significam, respectivamente, pegar (*get*) e atribuir (*set*).
O método *Set* serve para atribuir dados a um objeto.
O método *Get* é utilizado no momento de "resgatar" este mesmo dado atribuído. Exemplo:

```
class Cachorro:
    def __init__(self, nome, raca, idade):
        self.__nome = nome
        self.raca = raca
        self.idade = idade

    def getNome(self):
        return self.__nome

    def setNome(self, nome):
        self.__nome = nome
    #métodos getRaca() e setRaca() omitidos
    #métodos getIdade() e setIdade() omitidos
```

Devemos sempre acessar e modificar os atributos de uma classe utilizando os métodos `set` e `get`. Exemplo:

```
dog1 = Cachorro("Rex", "bulldog", 3)
print(dog1.getNome())
dog1.setNome("Bob")
print(dog1.getNome())
```

10. Propriedade __slots__

__slots__ é uma **variável** embutida dentro da nossa classe que guarda uma lista dos atributos da classe. Isso evita que novos atributos sejam criados em tempo de execução.

```
class Cachorro:
      __slots__ = 'nome','raca','idade'

      def __init__(self, nome, raca, idade):
          self.nome = nome
          self.raca = raca
          self.idade = idade
```

Se tentarmos acessar um atributo na classe *Cachorro* que não existe, então receberemos um erro. Exemplo:

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog1.sexo = "M"
```

AttributeError: 'Cachorro' object has no attribute 'sexo'

11. Alterando a visibilidade dos métodos de uma classe

Assim como os atributos, os métodos também podem se tornar privados. Um método privado significa que ele só pode ser acessado dentro da classe. Para tornar um método privado, utilizamos dois *underscores* ('__') antes do nome do método. Exemplo:

```
class Cachorro:
      __slots__ = 'nome','raca','idade'

      def __init__(self, nome, raca, idade):
          self.__nome = nome
          self.raca = raca
          self.idade = idade

      def __listarVacinas (self):
          print("vacinas em dia!")
```

No exemplo acima, o método *listarVacinas()* foi definido como privado. Isso significa que não conseguiremos acessá-lo fora da classe. Veja o exemplo a seguir:

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog1.__listarVacinas()
```

Ao executar o código acima, o interpretador irá acusar que o método *listarVacinas()* **não existe** na classe *Cachorro*. Somente dentro da classe conseguiremos acessar o método *listaVacinas()*. Veja o exemplo:

```
class Cachorro:
      __slots__ = 'nome','raca','idade'

      def __init__(self, nome, raca, idade):
          self.__nome = nome
          self.raca = raca
          self.idade = idade

      def __listarVacinas (self):
          print("vacinas em dia!")

      def verificarSaude(self):
          self.__listarVacinas()
```

Foi criado o método *verificarSaude()* que é público. Ele pode acessar o método *listarVacinas()* que é privado.

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog1.verificarSaude()
```

12. Métodos Estáticos

Nem sempre é necessário instanciar um objeto para utilizar seus métodos. Isto é feito usando o decorador **@staticmethod** antes da definição do método. Veja o exemplo.

```
class Cachorro:
    def __init__(self, nome, raca, idade):
        self.nome = nome
        self.raca = raca
        self.idade = idade

    def maior_idade(self, idade):
      if(idade >= 3):
        return True
      else:
        return False

    @staticmethod
    def valida_nome(nome):
        return len(nome) >= 3
```

Veja que não foi necessário passar *self* como parâmetro para o método *valida_nome()*

O método *valida_nome()* poderá ser chamado sem a necessidade de se instanciar um objeto do tipo Cachorro.

```
print(Cachorro.valida_nome('Lili'))
```

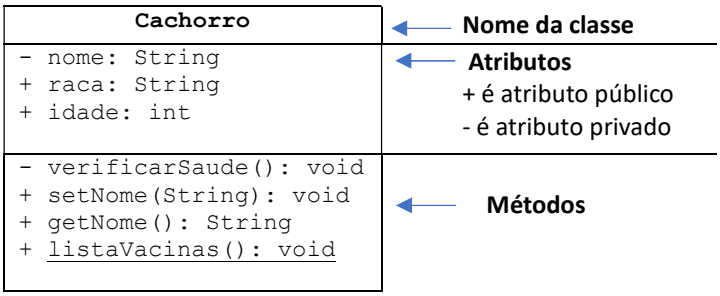
Entretanto, com o método *maior_idade()* isso não vai funcionar.

```
print(Cachorro.maior_idade(4))
```

Para utilizar o método *maior_idade()* devemos obrigatoriamente **instanciar** um objeto do tipo Cachorro. Veja o exemplo que irá funcionar:

```
dog = Cachorro("Rex", "bulldog", 3)
print(dog.maior_idade(5))
```

13. Representação UML de uma classe



Os métodos estáticos são sublinhados no diagrama. No exemplo acima o método *listaVacinas()* é estático