Guia para classes de Python

Resumo do *notebook* "Classes de Python"

Princípios fundamentais de classes, métodos e objetos em Python.

Escopo: Este documento é um resumo do *notebook* "Classes de Python" do módulo 2 do curso "Deep learning: domínio das redes neurais", do MIT xPRO. Sua finalidade é oferecer uma visão geral dos aspectos principais desse *notebook*.

Índice

1. Classes

O Python suporta programação orientada a objetos. De fato, tudo em Python é um objeto, e o programador também pode criar e usar novos tipos de objetos.

A programação orientada a objetos promove a modularização de dados e código, o que permite uma boa organização e poucos erros em caso de alterações feitas no código.

Definições:

- Um objeto é qualquer coisa que você deseja manipular ao trabalhar com código. Dependendo do tipo (classe) do objeto, pode haver atributos de dados associados.
- Uma classe é um modelo de código para criar e operar em um objeto.
- Uma instância é um objeto individual de uma classe.
- Um atributo de instância é uma variável que armazena dados ou propriedades para aquela instância específica.
- Um método é uma função que faz parte de uma classe e que normalmente opera em instâncias da classe.

1.a. Criação de classes

O *script* abaixo define uma classe que cria objetos do tipo Rectangle (retângulo). Observe que a indentação é importante.

A palavra-chave class indica que o código especifica ou cria uma classe. Os parênteses vazios () ao lado do nome da classe Rectangle indicam que essa classe não herda de outra classe. Posteriormente neste arquivo, há uma seção chamada "3. Herança" para ampliar o assunto.

```
class Rectangle():
    color = 'green'

    def_init_(self, length, width): self.length
        = length

        self.width = width
```

Ao criar uma instância (ou seja, instanciar uma instância de uma classe), o método inicializador _init_ é sempre chamado. O comportamento desejado para a inicialização da instância é especificado por def_init_() e define um método construtor.

! Importante: O argumento self frequentemente aparece nas definições de métodos com uma classe de Python e se refere à instância específica na qual o método está sendo chamado. Nesse caso de construtor, os atributos de instância length e width são específicos do objeto de instância, com valores definidos por esse construtor. Tenha em mente que, em geral, sempre que você chamar um método em uma instância posteriormente, o self não será um argumento explícito na chamada. Em vez disso, o Python organizará automaticamente para que o primeiro argumento do método (self) se refira àquela instância.

O exemplo abaixo instancia dois objetos da classe Rectangle.

```
# Instância 1 da classe
Rectangle
rectangle1 = Rectangle(7,8)

# Instância 2 da classe
Rectangle
rectangle2 = Rectangle(3,5)
```

1.b. Atributos de classes

Diferentemente dos atributos de instância, os atributos de classes são variáveis compartilhadas por todas as instâncias da classe (ou seja, essas variáveis e seus respectivos valores são armazenados na própria classe em vez de em cada instância). Na classe Rectangle definida antes, a cor verde, em inglês color = 'green', representa um atributo de classe que é constante em todos os objetos rectangle1 e rectangle2 dessa classe.

Podemos extrair e exibir esses atributos usando as convenções abaixo.

2.Métodos

Dentro de uma classe, o usuário pode definir métodos que operam em instâncias da classe. Os métodos podem ser considerados como funções que fazem parte de uma classe.

O script abaixo incorpora um método area para a classe Rectangle para calcular a área das instâncias de retângulos.

```
class Rectangle():
    color = 'green'

    def_init_(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width
```

2.a. Chamando um método

Para chamar um método, usamos um ponto após a instância para indicar qual método está sendo chamado e, em seguida, incluímos os parênteses () com quaisquer argumentos necessários para esse método.

No exemplo de código nesta demonstração, a sintaxe para chamar o método area seria a seguinte:

area = rectangle3.area()

O script abaixo define uma nova instância rectangle3 da classe Rectangle e chama o método area para calcular sua área.

```
# Instância 3 da classe
Rectangle
rectangle3 = Rectangle(9, 4)

# Chamar o método area nessa instância de
retângulo
area = rectangle3.area()
print("Rectangle Area:", area)

# Resultados de print:
Rectangle Area: 36
```

2.b. Métodos Getter e Setter

Ao definir classes, é útil definir métodos chamados "getters" ou "setters".

Definições:

- Um método **getter** fornece a capacidade de visualizar facilmente um atributo específico.
- Um método **setter** permite ao usuário atualizar o valor de um atributo específico.

O script abaixo incorpora um método setter chamado set_length para class Rectangle que permite ao usuário atualizar o valor da variável de comprimento length de uma instância de retângulo.

```
class Rectangle():
    color = 'green'

    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def set_length(self, new_length):
        self.length = new_length
```

O script a seguir utiliza a instância rectangle3 e chama o método set_length para atualizar o valor da variável length de 9 a 44.

Se o usuário imprimir os atributos da instância rectangle3, o valor de length será impresso como 44.

```
# Definir o comprimento rectangle3 com o método
set_length
rectangle3.set_length(44)

# rectangle3
print("Updated rectangle3 size: ", (rectangle3.length, rectangle3.width))

# Resultados de print:
Updated rectangle3 size: (44, 4)
```

3. Herança

A herança nos permite definir uma classe que herda métodos e atributos de outra classe.

Definições:

- Uma superclasse é uma classe da qual outras classes herdam atributos e métodos.
- Uma subclasse herda atributos e métodos de uma superclasse.

Atributos e métodos de uma superclasse agora também serão acessíveis na subclasse.

O script abaixo cria uma classe Square como uma subclasse da superclasse Rectangle:

```
class Square(Rectangle):

    def __init__(self, side):
        self.side = side
        Rectangle.__init__(self, side, side)
```

Importante: Se uma subclasse tiver seu próprio método def __init__(), ela não herda automaticamente ou executa o método __init__() da superclasse. Portanto, o construtor __init__() da subclasse substitui o construtor __init__() da superclasse.

No exemplo acima, a subclasse Square define o método __init__(self, side), em que side é o atributo ou variável de instância para essa subclasse.

No entanto, o construtor da superclasse também é chamado pelo código:

Rectangle.__init__(self, side, side)

Esta linha de código indica que qualquer valor da variável side da subclasse Square é atribuído aos atributos length e width da superclasse. Em outras palavras, ao definir um objeto com a classe Square, seus atributos também são transferidos para as variáveis da superclasse Rectangle de forma consistente.