Transcrição do curso de Python – Brasilidades

Nesta aula construímos uma classe que verifica se o CPF inserido pelo usuário é valido ou não. Incialmente o documento foi validado somente pela quantidade de caracteres e em seguida, por meio de uma pacote que baixamos no [PyPi(Python Packege Index)](https://pypi.org/). A classe abaixo deve verificar o CPF inserido no momento que é instanciada:

class ValidaCpf:

def \_\_init\_\_(self, documento):

self.analisa\_cpf(documento)

def analisa\_cpf(self, documento):

if len(documento) == 11:

valida\_cpf = CPF()

if valida\_cpf.validade(documento):

self.cpf = documento

print('CPF salvo')

else:

print('CPF inserido não é valido')

else:

print('Número de digitos incorreto')

cpf\_tres = '12354367912'

objeto\_cpf\_tres = CPF(cpf\_tres )

É importante que você esteja habituado a ler códigos de outras pessoas (tarefa comum no seu dia a dia de programador). Pensando nisso quero propor que você de uma olhada na classe e nos métodos que nos utilizamos durante nossa aula. Abra o [Repositório](https://github.com/alvarofpp/validate-docbr/blob/master/validate_docbr/CPF.py) da biblioteca que utilizamos e divirta-se!

Não se preocupe em entender perfeitamente como tudo está funcionando, mas é muito interessante que você dê uma lida no código e, caso você não tenha familiaridade com a plataforma GitHub, aproveite este momento para criar uma conta e ir se acostumando, pois no trabalho com programação você vai passar muito tempo no GitHub! E dê uma estrelinha lá no código para incentivar e dar o reconhecimento merecido para quem o fez!

Nesta aula, criamos uma classe responsável por trabalhar com CPF e, para isso, utilizamos uma classe baixada do PyPI. Vamos refazer os passos dessa instalação e criar um método que utilize a biblioteca validate\_docbr para formatar CPFs.

É preciso acessar a página da biblioteca [dentro do PyPI](https://pypi.org/project/validate-docbr/).

1. Devemos instalar a biblioteca com o pip install.

pip install validate-docbr

1. Agora podemos construir o código, sem esquecer de importar a classe necessária da biblioteca.

from validate\_docbr import CPF

1. Então, vamos construir a classe e o método que vão servir para criar a máscara.

from validate\_docbr import CPF

class ValidaCpf:

def format\_cpf(self):

cpf\_mask = CPF()

return cpf\_mask.mask(self.cpf)

1. Agora, vamos jogar esse método dentro da representação string da classe.

def \_\_str\_\_(self)

return self.format\_cpf()

Pronto! Tente fazer na sua máquina e, caso tenha alguma dúvida, dê uma olhada nos vídeos da aula. Se a dúvida persistir, pergunte no fórum do curso.

Nessa aula, entendemos como:

* Validar um documento pela quantidade de caracteres;
* Encontrar, instalar e importar bibliotecas no PyPI;
* Ler documentações de bibliotecas e utilizá-las em seus códigos.

Na próxima aula, vamos continuar construindo nossa classe e fazer com que ela trabalhe com CNPJs também! Vejo vocês lá!

O código do arquivo cpf.py será:

from validate\_docbr import CPF  
  
  
class Cpf:  
 def \_\_init\_\_(self, documento):  
 documento = str(documento)  
 if self.cpf\_eh\_valido(documento):  
 self.cpf = documento  
 else:  
 raise ValueError("CPF inválido!")  
  
  
 def cpf\_eh\_valido(self, documento):  
 if len(documento) == 11:  
 valida\_cpf = CPF() # cria um objeto importado da biblioteca para a validação  
 return valida\_cpf.validate(documento) # retorna um boleano  
 else:  
 raise ValueError("Quantidade de dígitos inválida")  
  
 def formata\_cpf(self):  
 mascara = CPF()  
 return mascara.mask(self.cpf)  
  
 def \_\_str\_\_(self):  
 return self.formata\_cpf()

Como vimos durante a aula, o padrão de projeto Factory pode ser bastante útil em situações em que temos objetos que pertencem a classes semelhantes. É importante perceber que os métodos das subclasses da classe factory possuem os métodos com os nomes iguais! Dessa forma, podemos usar a factory sem se preocupar em qual classe foi instanciada.

Esta explicação é uma simplificação desse padrão de projeto. Abaixo listei alguns artigos com explicações mais formais.

Artigos em português:

* [Artigo 1](https://pt.wikipedia.org/wiki/Factory_Method)
* [Artigo 2](https://www.thiengo.com.br/padrao-de-projeto-factory-method) Artigos em inglês:
* [Artigo 3](https://sourcemaking.com/design_patterns/factory_method) (Site incrível!)
* [Artigo 4](https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Factory.html)

Os artigos estão ordenados do mais simples ao mais complexo. Recomendo a leitura de todos, pois esse padrão de projeto é bem comum no código de diversos sistemas.

Vamos fazer um método factory juntos! Para criar um padrão factory, precisamos entender com o que estamos trabalhando e como vamos saber qual a hora certa de chamar uma classe ou outra. Vamos utilizar o mesmo exemplo da aula e trabalhar com documentos.

1. Vamos decidir como vamos selecionar uma classe ou outra:

A classe CPF será chamada quando o documento conter 11 dígitos, e a classe CNPJ quando possuir 14.

1. Criamos a classe e definimos um método estático que decidirá qual classe instanciar:

class Documento:

@staticmethod

def criar\_novo(documento):

1. Agora, criamos os condicionais:

class Documento:

@staticmethod

def criar\_novo(documento):

doc\_str = str(documento)

if len(doc\_str) == 11:

return DpcCpf(doc\_str)

if len(doc\_str) == 14:

return DpcCnpj(doc\_str)

1. Depois disso, fica fácil incluir uma opção nova de classe:

class Documento:

@staticmethod

def criar\_novo(documento):

doc\_str = str(documento)

if len(doc\_str) == 11:

return DpcCpf(doc\_str)

if len(doc\_str) == 14:

return DpcCnpj(doc\_str)

if len(doc\_str) == 20:

return DocQualquer(doc\_str)

1. Por último, vamos configurar uma exceção para quando o documento não for de nenhum tipo:

class Documento:

@staticmethod

def criar\_novo(documento):

doc\_str = str(documento)

if len(doc\_str) == 11:

return DpcCpf(doc\_str)

if len(doc\_str) == 14:

return DpcCnpj(doc\_str)

if len(doc\_str) == 20:

return DocQualquer(doc\_str)

Raise ValueError(“Documento incorreto!”)

Depois o código ficou

from validate\_docbr import CPF, CNPJ  
  
  
class Documento:  
 @staticmethod  
 def cria\_documento(documento):  
 documento = str(documento)  
 if len(documento) == 11:  
 return DocCpf(documento)  
 elif len(documento) == 14:  
 return DocCnpj(documento)  
 else:  
 raise ValueError("Documento incorreto!")  
  
  
class DocCpf:  
 def \_\_init\_\_(self, documento):  
 if self.valida(documento):  
 self.cpf = documento  
 else:  
 raise ValueError("CPF inválido")  
  
 def \_\_str\_\_(self):  
 return self.formata()  
  
 def valida(self, documento):  
 validador = CPF()  
 return validador.validate(documento)  
  
 def formata(self):  
 mascara = CPF()  
 return mascara.mask(self.cpf)  
  
  
class DocCnpj:  
 def \_\_init\_\_(self, documento):  
 if self.valida(documento):  
 self.cnpj = documento  
 else:  
 raise ValueError("CNPJ inválido")  
  
 def \_\_str\_\_(self):  
 return self.formata()  
  
 def valida(self, documento):  
 validador = CNPJ()  
 return validador.validate(documento)  
  
 def formata(self):  
 mascara = CNPJ()  
 return mascara.mask(self.cnpj)

e para chamar a classe ficou:

from cpf\_e\_cnpf import Documento  
  
cpf = Documento.cria\_documento(10655092781)  
  
print(cpf)  
  
cnpj = Documento.cria\_documento(28176998000441)  
  
print(cnpj)

Nessa aula, entendemos:

* Como utilizar mais uma classe da validate\_docbr;
* O que é o padrão de projeto Factory;
* Como e quando implementar uma Factory em nosso código.

Na próxima aula, vamos construir mais uma classe em nossa biblioteca, e dessa vez ela irá trabalhar com números de telefone e e-mails.