aula_04

September 30, 2019

Universidade Federal de Roriama Departamento de Ciência da Computação Professor: Filipe Dwan Pereira Código da disciplina: DCC305 Período: 2019.2

0.1 Disclaimer

Esta aula é uma adaptação do capítulo 4 do livro:

- Phillips, Dusty. Python 3 Object-oriented Programming - Unleash the power of Python 3 objects. "Packt Publishing", 2015. Second Edition.

Nesta aula aprenderemos:

- Como lançar exceções;
- Como se recuperar de uma exceção lançada;
- Como lidar com diferentes exceções de maneiras diferentes;
- Criar novos tipos de exceções;
- Usar a sintaxe de exceções para um fluxo de controle;

0.2 Introdução

- Seria ideal se o código sempre retornasse um resultado válido, mas às vezes um resultado válido não pode ser calculado.
  - Por exemplo, não é possível dividir por zero ou acessar o oitavo item em uma lista de cinco itens.
- Antigamente, a única maneira de contornar isso era verificar rigorosamente as entradas de todas as funções para garantir que elas fizessem sentido.
- Tipicamente, funções tem valores de retorno especiais para indicar uma condição de erro;
  - por exemplo, eles poderiam retornar um número negativo para indicar que um valor positivo não pôde ser calculado;
  - Números diferentes podem significar erros diferentes;
  - Qualquer código que chamasse essa função teria que verificar explicitamente uma condição de erro e agir em conformidade.
  - Muito código não se deu ao trabalho de fazer isso, e programas simplesmente falharam;
- Em programação orientada a objetos usamos o conceito de exceptions, um tipo especial de objeto que é manipulado quando faz sentido manipulá-lo;
- As exceções são objetos especiais tratados dentro do fluxo de controle do programa;

0.3 Um pouco sobre exceções no python

- Uma exceção é um objeto;
- Existem várias classes de exceções diferentes;
- Todas as classes herdam da classe **BaseException**;

Para ilustrar, veja um exemplo em que uma exceção é lançada, onde iremos tentar imprimir uma string sem usar os parentêses (estamos usando o python 3):

[1]:
```
print "hello world"
```

```
  File "<ipython-input-1-6d29d8fb337c>", line 1
    print "hello world"
                       ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello␣
↪world")?
```

- Sempre que o python se depara com uma linha do seu programa que ele não consegue entender, então é lançado um SyntaxError, que é um tipo de exceção;
- Veja outros exemplos de exceções:

[2]:
```
x = 5/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)

<ipython-input-2-fc2abf138dd5> in <module>
----> 1 x = 5/0


ZeroDivisionError: division by zero
```

[9]:
```
lista = range(5)
print(lista[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

<ipython-input-9-f38cd5df3ef1> in <module>
  1 lista = range(5)
```

```
----> 2 print(lista[10])


    IndexError: range object index out of range
```

[10]:
```
lista + 2.34
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-10-42ce0bfd3c54> in <module>
----> 1 lista + 2.34


    TypeError: unsupported operand type(s) for +: 'range' and 'float'
```

[11]:
```
lista.adiciona
```

```
    ---------------------------------------------------------------------------

    AttributeError                            Traceback (most recent call last)

    <ipython-input-11-618bc9227df2> in <module>
----> 1 lista.adiciona


    AttributeError: 'range' object has no attribute 'adiciona'
```

[12]:
```
d = {'1': 'um'}
d['2']
```

```
    ---------------------------------------------------------------------------

    KeyError                                  Traceback (most recent call last)

    <ipython-input-12-88d0c6fdc283> in <module>
      1 d = {'1': 'um'}
----> 2 d['2']
```

```
KeyError: '2'
```

[13]:
```
print(variavel_nao_inicializada)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-13-553228d19fbc> in <module>
----> 1 print(variavel_nao_inicializada)


NameError: name 'variavel_nao_inicializada' is not defined
```

- Note que as exceções acima são indicativos de que nosso programa está com erro, assim sendo, é importante que as evitemos programaticamente;

0.4 Lançando exceções

- Podemos usar o mesmo mecanimos que o python utiliza para lançar exceções;
- Veja abaixo um exemplo da classe EvenOnly que é uma lista que só armazena valores inteiros e pares:
  - Lançaremos uma exceção de tipo, caso o usuário tente adicionar um item diferente de inteiro;
  - Lançaremos uma exceção de valor inválido, caso o usário tente adicionar um inteiro não par;

[1]:
```
class EvenOnly(list):
    def append(self, valor):
        if not isinstance(valor, int):
            raise TypeError("Somente inteiros podem ser adicionados")
        if valor%2==1:
            raise ValueError("Somente números pares podem ser adicionados")
        super().append(valor)
```

[4]:
```
lista = EvenOnly()
lista.append("teste")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
    <ipython-input-4-9af90eda81e1> in <module>
      1 lista = EvenOnly()
----> 2 lista.append("teste")


    <ipython-input-2-23bbdf985b95> in append(self, valor)
      2     def append(self, valor):
      3         if not isinstance(valor, int):
----> 4             raise TypeError("Somente inteiros podem ser adicionados")
      5         if valor%2==1:
      6             raise ValueError("Somente números pares podem ser␣
↪adicionados")


    TypeError: Somente inteiros podem ser adicionados
```

- Perceba que no exemplo acima é mostrado a linha onde ocorreu o erro e o tipo de erro;
- Agora Vamos tentar adicionar um valor não inteiro na nossa lista:

[5]:
```
lista.append(1)
```

```
    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-5-1f651333501d> in <module>
----> 1 lista.append(1)


    <ipython-input-2-23bbdf985b95> in append(self, valor)
      4             raise TypeError("Somente inteiros podem ser adicionados")
      5         if valor%2==1:
----> 6             raise ValueError("Somente números pares podem ser␣
↪adicionados")
      7         super().append(valor)


    ValueError: Somente números pares podem ser adicionados
```

- Por fim, vamos testar um exemplo que funciona:

[6]:
```
lista.append(2)
lista.append(4)
lista.append(6)
lista
```

[6]:
```
[2, 4, 6]
```

0.5 O efeito de uma exceção

- Quando uma exceção é lançada, ao menos que o programa trate essa exceção, execução dele seré interrompida imadiatamente;
- Veja o exemplo abaixo, onde não há tratamento:

[7]:
```
def no_return():
    print("Eu estou prestes a lançar uma exceção")
    raise Exception("Essa exceção é sempre lançada")
    print("Essa linha nunca será executada.")
    return "Nunca a função retornará nada!"
```

- Se executarmos a função, as linhas 4 e 5 nunca serão executadas:

[8]:
```
no_return()
```

```
Eu estou prestes a lançar uma exceção


        ---------------------------------------------------------------------------

        Exception                                 Traceback (most recent call last)

        <ipython-input-8-7cb40636301c> in <module>
    ----> 1 no_return()


        <ipython-input-7-661bce91b873> in no_return()
          1 def no_return():
          2     print("Eu estou prestes a lançar uma exceção")
    ----> 3     raise Exception("Essa exceção é sempre lançada")
          4     print("Essa linha nunca será executada.")
          5     return "Nunca a função retornará nada!"


        Exception: Essa exceção é sempre lançada
```

- Note que se você tem uma função que chama outra função que lança exceção, a primeira não executará depois do ponto em que a segunda função é chamada.
- Veja o exemplo:

[9]:
```
def call_exceptor():
    print("chama uma função que lança exceção...")
    no_return()
```

```
    print("uma exceção foi lançada...")
    print("...essas linhas não serão executadas")
```

- Veja abaixo o trabeck (saída da exceção);
- Observe que como a exceção interrompe a execução do programa porque ela não é tratada nem no *call_exceptor* nem no *no_retorn*:

[10]:
```
call_exceptor()
```

```
chama uma função que lança exceção...
Eu estou prestes a lançar uma exceção


        ---------------------------------------------------------------------------

        Exception                                 Traceback (most recent call last)

        <ipython-input-10-e86660b7de9c> in <module>
    ----> 1 call_exceptor()


        <ipython-input-9-853da568a806> in call_exceptor()
          1 def call_exceptor():
          2     print("chama uma função que lança exceção...")
    ----> 3     no_return()
          4     print("uma exceção foi lançada...")
          5     print("...essas linhas não serão executadas")


        <ipython-input-7-661bce91b873> in no_return()
          1 def no_return():
          2     print("Eu estou prestes a lançar uma exceção")
    ----> 3     raise Exception("Essa exceção é sempre lançada")
          4     print("Essa linha nunca será executada.")
          5     return "Nunca a função retornará nada!"


        Exception: Essa exceção é sempre lançada
```

0.6 Tratando Exceções

- Agora entenderemos como nos recuperar de uma exceção;
- Para tanto, usaremos a cláusula try…except, isto é, tente executar um código perigoso e se ocorrer uma exceção capture-a e trate-a;
- Vejamos um exemplo disso com o método no_retorn supracitado:

[11]:
```
try:
    no_return()
except:
    print("Exceção capturada")
print("Execução normal após exceção capturada")
```

```
Eu estou prestes a lançar uma exceção
Exceção capturada
Execução normal após exceção capturada
```

- O problema com o código acima é que ele vai capturar qualquer tipo de exceção;
- Por exemplo, imagine que estamos escrevendo um códido que pode lançar erro por divisão por zero e erro de tipagem;
  - Imagine ainda que queremos tratar o primeiro erro, mas o segundo nós queremos que seja apresentado, caso ocorra;
- Para capturar exceções específicas, usamos a seguinte sintaxe:

[13]:
```
def funny_division(divider):
    try:
        return 100 / divider
    except ZeroDivisionError:
        return "Zero is not a good idea!"
```

[14]:
```
print(funny_division(0))
```

```
Zero is not a good idea!
```

[15]:
```
print(funny_division(50.0))
```

```
2.0
```

[17]:
```
print(funny_division("hello"))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-17-87a168536dac> in <module>
----> 1 print(funny_division("hello"))


<ipython-input-13-d740deb7abb2> in funny_division(divider)
  1 def funny_division(divider):
  2     try:
```

```
----> 3         return 100 / divider
      4     except ZeroDivisionError:
      5         return "Zero is not a good idea!"


TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

[18]:
```
print(funny_division())
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-18-66ebf83656ea> in <module>
----> 1 print(funny_division())


TypeError: funny_division() missing 1 required positional argument:␣
↪'divider'
```

- Alternativamente, podemos capturar duas ou mais exceções, conforme segue:

[8]:
```
def funny_division2(anumber):
    try:
        if anumber == 13:
            raise ValueError("13 é um número bloqueado")
        return 100 / anumber
    except ZeroDivisionError:
        return "Entre com um número diferente de zero"
    except TypeError:
        return "Entre com um valor numérico"
    except ValueError:
        print("13 não!")
        raise#lança novamente a exceção ValueError
```

- A palavra raise na linha 12 lança novamente a exceção após capturá-la;
- Veja um exemplo:

[2]:
```
funny_division2(13)
```

```
13 não!
```

```
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)

<ipython-input-2-ac43266bf1a1> in <module>
----> 1 funny_division2(13)


<ipython-input-1-a3cbe219affb> in funny_division2(anumber)
      2     try:
      3         if anumber == 13:
----> 4             raise ValueError("13 é um número bloqueado")
      5         return 100 / anumber
      6     except ZeroDivisionError:


ValueError: 13 é um número bloqueado
```

- Observe que a ordem de tratamento das exceções é importante;
- Por exemplo, se tratássemos primeiramente a exceção Exception, nunca as outras seriam executadas, uma vez que todas as outras exceções herdam de Exception e, logo, elas são uma Exception;

Outra opção é capturar qualquer uma dessas exceções e imprimir o objeto da exceção:

[3]:
```
def funny_division2(anumber):
    try:
        if anumber == 13:
            raise ValueError("13 é um número bloqueado")
        return 100 / anumber
    except (ZeroDivisionError, TypeError, ValueError) as e:
        print("Erro:", e)
```

[4]:
```
funny_division2(13)
```

```
Erro: 13 é um número bloqueado
```

[5]:
```
funny_division2("string")
```

```
Erro: unsupported operand type(s) for /: 'int' and 'str'
```

[6]:
```
funny_division2(0)
```

```
Erro: division by zero
```

Outra opção é imprimir os argumentos que foram passados dentro da exceção através do atributo *args*:

[7]:
```
try:
    raise ValueError("Esse é um argumento", 'Outro argumento', 1)
except ValueError as e:
    print("Os argumentos da exceção foram: ", e.args)
```

```
Os argumentos da exceção foram:  ('Esse é um argumento', 'Outro argumento', 1)
```

- Existem ainda opções para executar código independente se a exceção ocorreu ou não.
- Para tanto, existem as palavras-chave *finally* e *else*:
  - A primeira é sempre executada;
  - a última é executada caso não ocorra nenhuma exceção;
- Veja o exemplo abaixo, onde lançamos uma exceção aleatoriamente:

[9]:
```
import random
def exemplo_excecoes_aleatorias():
    some_exceptions = [ValueError, TypeError, IndexError, None]
    try:
        choice = random.choice(some_exceptions)
        print("raising {}".format(choice))
        if choice:
            raise choice("An error")
    except ValueError:
        print("ValueError Capturado")
    except TypeError:
        print("TypeError Capturado")
    except Exception as e:
        print("Um outro tipo de erro capturado: %s" % ( e.__class__.__name__))
    else:
        print("Esse código é chamado se não houver nenhuma exceção")
    finally:
        print("Esse código é sempre chamado (ainda que haja erro)")
```

[42]:
```
exemplo_excecoes_aleatorias()
```

```
raising None
Esse código é chamado se não houver nenhuma exceção
Esse código é sempre chamado (ainda que haja erro)
```

[43]:
```
exemplo_excecoes_aleatorias()
```

```
raising <class 'TypeError'>
TypeError Capturado
Esse código é sempre chamado (ainda que haja erro)
```

[46]:
```
exemplo_excecoes_aleatorias()
```

```
raising <class 'IndexError'>
Um outro tipo de erro capturado: IndexError
Esse código é sempre chamado (ainda que haja erro)
```

- Alguns exemplos de uso da palavra-chave *finally*:
  - Fechar uma conexão com o banco de dados;
  - Fechar um arquivo;
  - Fechar uma conexão qualquer e etc.
- Alerta:
  - Cuidado quando nenhuma exceção é capturada, pois as cláusulas *else* e *finally* são ambas executadas;
  - Qualquer uma das cláusulas *except*, *else* e *finally* podem ser omitidas após um bloco try
    - Note que o *else* é uma cláusula opcional, que quando presente, deve vir depois de todas as cláusulas except.
  - Caso você queira usar tadas as palavras-chave, a ordem deve ser: try, else, finally;
  - A ordem das exceções são da mais específica para a mais genérica;
- Apenas para consolidar o uso do *else*, veja o exemplo abaixo extraído da documentação do python:

[9]:
```
%%file myfile.txt
Poo com python é bem legal.
Boa vista é uma cidade muito bonita.
O Brasil é um lindo país.
```

```
Writing myfile.txt
```

[13]:
```
try:
    f = open('myfile.txt', 'r')
except OSError:
    print('Arquivo não pode ser aberto', arg)
else:
    print('o aquivo tem', len(f.readlines()), 'linhas')
    f.close()
```

```
o aquivo tem 3 linhas
```

- Perceba que o conteúdo dentro do else só será executado se não for lançada uma exceção *OSError*;
  - Caso não haja um essa exceção, aí sim podemos imprimir a quantidade de linhas do arquivo e fechá-lo (linhas 7 e 8).

0.6.1 Nota

Alternativa, você pode usar a palavra-chave **with**, que permite usar objetos como arquivos de maneira segura e sem precisar fechar o arquivo. Veja:

[ ]:
```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

0.7 Hierarquia das exceções

- A maioria das exceções herdam da classe *Exception* (mas não todas);
- Todas as exceções do python herdam da classe *BaseException* direta ou indiretamente;
  - A classe *Exception* herda da classe *BaseException*;
- Existem duas exceções chaves, *SystemExit* e *KeyboardInterrupt*, que herdam diretamente de *BaseException*, ao invés de *Exception*.
  - A *SystemExit* é lançada quando o programa acaba naturalmente, tipacamente porque foi chamada a função sys.exit (e.g. quando escolhemos a opção sair do menu);
  - A *KeyboardInterrupt* é normalmente lançada em programas de linhas de comando do python. Ela é lançada quando o usuário interrompe a execução do programa, por exemplo, quando pressiona Ctrl+C.
- Veja abaixo o diagrama de classes que ilustra a hierarquia das exceções no python:
- Note que quando usamos a cláusula **except** sem argumentos (sem especificar a classe), nós capturamos todas as exceções, incluindo essas duas exceções especiais;
  - Essas exceções especiais normalmente precisam de tratamento especial (seja para salvar um arquivo, fechar uma conexão ou algo do tipo no bloco finally);
- Assim, não vale a pena usarmos a cláusula **except** sem argumentos;
  - Se quisermos capturar qualquer exceção diferente de *SystemExit* e *KeyboardInterrupt*, explicitamos Exception na cláusura except (except Exception);

0.8 Criando nossas próprias exceções

- Muitas vezes ocorre de você querer lançar uma exceção, mas nenhuma das opções built-in são viáveis.
- Para criar nossas próprias exceções basta herda de um exceção, veja:

[2]:
```
class InvalidWithdrawal(Exception):
    pass
```

- Para que o erro fique mais fácil de tratar é importante que o nome da classe seja algo significativo.
- Abaixo vamos lançar a exceção criada:

[3]:
```
raise InvalidWithdrawal('Você não tem R$ 200 na sua conta')
```

```
---------------------------------------------------------------------------

InvalidWithdrawal                         Traceback (most recent call last)

<ipython-input-3-35092f2524d1> in <module>
----> 1 raise InvalidWithdrawal('Você não tem R$ 200 na sua conta')


InvalidWithdrawal: Você não tem R$ 200 na sua conta
```

- No caso acima, foi passada uma string para o construtor da exceção.
- Lembre-se que podemos passar um número arbitrário de argumentos para a exceção, os quais serão guardados em uma tupla da classe Exception chamada **args**.
  - Isso faz com que ficque fácil criar exceções em python, sem precisar sobreescrever o __init__ da classe *Exception*;
- Entretanto, caso você deseje customizar o __init__, você pode:

[26]:
```
class InvalidWithdrawal(Exception):
    def __init__(self, balance, amount):
        super().__init__("Nao foi possivel realizar o saque: "
            "a conta não tem R${}".format(amount))
        self.amount = amount
        self.balance = balance
    def valor_faltante(self):
        return self.amount - self.balance
```

[15]:
```
raise InvalidWithdrawal(25, 200)
```

```
---------------------------------------------------------------------------

InvalidWithdrawal                         Traceback (most recent call last)

<ipython-input-15-74b59c4772bc> in <module>
----> 1 raise InvalidWithdrawal(25, 200)


InvalidWithdrawal: Nao foi possivel realizar o saque: a conta não tem R$200
```

- Podemos dar uma mensagem usando o método valor_faltante:

[16]:
```
try:
    raise InvalidWithdrawal(25, 50)
except InvalidWithdrawal as e:
```

```
    print("O valor que você está tentando sacar "
    "é maior que o seu saldo. Esta faltando o seguinte valor: "
    "${}".format(e.valor_faltante()))
```

```
O valor que você está tentando sacar é maior que o seu saldo. Esta faltando o
seguinte valor: $25
```

- Podemos usar a exceção *InvalidWithdrawal* no nosso exemplo de ATM da aula passada:

[27]:
```
class ATM:
    def realizar_operacao(self, operacao):
        try:
            operacao.realizar()
        except InvalidWithdrawal as e:
            print("Erro: ", e)
        except Exception as e:
            print("Erro: ", e)

class Conta:
    def __init__(self, agencia, numero):
        self.agencia = agencia
        self.numero = numero
        self.saldo = 0.0

    def sacar(self, valor):
        if valor>self.saldo or valor<0.0:
            raise InvalidWithdrawal(self.saldo, valor)
        self.saldo -= valor

    def depositar(self, valor):
        if valor<0.0:
            raise ValueError("Depósito:: Valor Negativo")
        self.saldo += valor

class Transferencia:
    def __init__(self, conta_origem, conta_destino, valor):
        self.conta_origem = conta_origem
        self.conta_destino = conta_destino
        self.valor = valor

    def realizar(self):
        self.conta_origem.sacar(self.valor)
        self.conta_destino.depositar(self.valor)

class Deposito:
    def __init__(self, conta_destino, valor):
        self.conta_destino = conta_destino
```

```
        self.valor = valor

    def realizar(self):
        self.conta_destino.depositar(self.valor)

class Saque:
    def __init__(self, conta, valor):
        self.conta = conta
        self.valor = valor

    def realizar(self):
        self.conta.sacar(self.valor)
```

[28]:
```
atm = ATM()
```

[29]:
```
c1 = Conta("2121-3", "31314-0")
c1.depositar(100)
c2 = Conta("2121-3", "31314-0")
c2.depositar(200)
```

[30]:
```
operacao_saque = Saque(c1, 150)
atm.realizar_operacao(operacao_saque)
```

```
Erro:  Nao foi possivel realizar o saque: a conta não tem R$150
```

0.9 Caso de Uso

- O caso de uso abaixo ajudará a consolidarmos conceitos aprendidos nesta aula e em aulas anteriores;
- Vamos criar um sistema de autenticação e um sistema de autorização;
  - Lembrando que autenticação é o processo de assegurar que um usuário é realmente a pessoa que ele diz que é;
  - Autorização é a verificação se um dado usuário (autenticado) pode executar determinada ação;
- O sistema como um todo será organizado em um único módulo;
- Como nosso intuito é consolidar o conteúdo aprendido, nossos sistemas provavelmente terão alguns (muitos) furos de segurança;
- Fazendo uma rápida análise orientada a objetos, o sistema de autorização funcionará de forma simples:
  - Criaremos uma lista de permissões que guarda usuários específicos que podem realizar ações;
  - Criaremos também algumas características administrativas para permitir que novos usuários sejam adicionados no sistema;
- Pensendo agora no design orientado a objetos, precisaremos de uma classe *User* que terá como atributos um *username* e um *password*;

- Criaremos ainda uma classe para ser a central de autencitação (*Authenticator*), que gerenciar os logins e logouts;
- Teremos ainda a classe *Authorizor* que será responsável por gerenciar as permissões dos usários para executar determinadas ações do sistema;
- Iremos providenciar apenas uma instância dessas classes no módulo *auth*, assim outros módulos podem usar esse mecanismos central para todas as suas autenticações e autorizações;
  - Caso eles queiram instâncias privadas dessas classes (atividades de autorização não-centrais), eles poderão fazer isso também;
- Em relação às exceções, primeiramente vamos definir uma exceção chamada AuthException para ser lançada em casos de problemas de autenticação;

- Agora vamos pensar na programação orientada a objetos.
- Vamos começar pela classe User, que será inicializado com *username* e *password*;
- O *password* será armazenado de modo criptografado;
- Precisaremos de um método *check_password* para verificar se a senha que o usuário passou está correta.
- Vejamos como ficará a classe User:

[21]:
```
import hashlib

class User:
    def __init__(self, username, password):
        '''Cria um novo usuário. A senha é
        criptografada depois que é salva.'''
        self.username = username
        self.password = self._encrypt_pw(password)
        self.is_logged_in = False
    def _encrypt_pw(self, password):
        '''Criptografa o password e depois retorna o sha.'''
        hash_string = (self.username + password)
        hash_string = hash_string.encode("utf8")
        return hashlib.sha256(hash_string).hexdigest()
    def check_password(self, password):
        '''Returna True se a senha for válida para
        este usuário, do contrário retorna False'''
        encrypted = self._encrypt_pw(password)
        return encrypted == self.password
```

- Como de prache, vamos testar nossa clase User:

[22]:
```
u = User('filipe', '123456')
```

[23]:
```
u.check_password('123456')
```

[23]:
```
True
```

[24]:
```
u.check_password('12345')
```

[24]:
```
False
```

- Em relação à classe *Authenticator*, toda a vez que um usuário for criado ele será adicionado em um dicionário dessa classe;
- Caso o usuário já exista no dicionário, a exceção *UsernameAlreadyExists* será lançada;
- Além disso, por razões de segurança, a exceção *PasswordTooShort* será lançada se o usuário criar um senha muito curta;
- Ambas exceções serão filhas da já mencionada *AuthException*;
- Assim sendo, antes de criar a classe *Authenticator*, vamos criar essas exceções:

[25]:
```
class AuthException(Exception):
    def __init__(self, username, user=None):
        super().__init__(username, user)
        self.username = username
        self.user = user

class UsernameAlreadyExists(AuthException):
    pass

class PasswordTooShort(AuthException):
    pass
```

- Agora podemos criar a classe *Authenticator*;
- Essa classe deve ter um dicionário que mapeia *usernames* para instâncias da classe *User*;
- O método que adiciona usuários deve checar as condições mencionadas antes de inseri-los no dicionário.
- A classe vai ficar assim:

[39]:
```
class Authenticator:
    def __init__(self):
        '''Construtor de um autenticador que gerencia
        logins e logouts de usuários.'''
        self.users = {}
    def add_user(self, username, password):
        if username in self.users:
            raise UsernameAlreadyExists(username)
        if len(password) < 6:
            raise PasswordTooShort(username)
        self.users[username] = User(username, password)
```

- Precisamos ainda de um método para realizar o login;
- Veja abaixo as possíveis exceções que podem ser lançadas no login:

[40]:
```
class InvalidUsername(AuthException):
    pass
class InvalidPassword(AuthException):
    pass
```

- Agora podemos criar nosso login na classe *Authenticator*:

[ ]:
```
def login(self, username, password):
    try:
        user = self.users[username]
    except KeyError:
        raise InvalidUsername(username)
    if not user.check_password(password):
        raise InvalidPassword(username, user)
    user.is_logged_in = True
    return True
```

- Observe que manipulamos KeyError, caso o username não seja uma das chaves do dicionário;
- Podemos ainda criar um método para verificar se um usuário está *logged in*:

[41]:
```
def is_logged_in(self, username):
    if username in self.users:
        return self.users[username].is_logged_in
    return False
```

- Finalmente, deixaremos um objeto authenticator a nível de módulo, assim o cliente pode acessá-lo como *auth.authenticator*:

[ ]:
```
authenticator = Authenticator()
```

- Agora podemos criar nossa classe *Authorizor*;
- Essa classe não deve dar autorização se o usuário não estiver *logged in*;
- Precisaremos configurar um dicionário com as devidas permissões de usuários:
- Além disso, iremos criar exceções para casos específicos;
- Veja como vão ficar nossas classes:

[26]:
```
class NotLoggedInError(AuthException):
    pass

class NotPermittedError(AuthException):
    pass

class PermissionError(Exception):
    pass

class Authorizor:
    def __init__(self, authenticator):
        self.authenticator = authenticator
        self.permissions = {}

    def add_permission(self, perm_name):
```

```
        '''Crie uma nova permissão à qual os usuários
        possam ser adicionados'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            self.permissions[perm_name] = set()
        else:
            raise PermissionError("Essa permissao ja Existe")

    def permit_user(self, perm_name, username):
        '''Concede permissão ao usuário'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Sem permissão")#ATUALIZAR str
        else:
            if username not in self.authenticator.users:
                raise InvalidUsername(username)
            perm_set.add(username)
```

- Finalmente, iremos deixar uma instância da classe a nível de módulo, conforme fizemos no authenticator:

[ ]:
```
authorizor = Authorizer(authenticator)
```

- Abaixo vamos colocar tudo junto no nosso módulo auth:

[1]:
```
%%file auth.py

import hashlib

class User:
    def __init__(self, username, password):
        '''Cria um novo usuário. A senha é
        criptografada depois que é salva.'''
        self.username = username
        self.password = self._encrypt_pw(password)
        self.is_logged_in = False
    def _encrypt_pw(self, password):
        '''Criptografa o password e depois retorna o sha.'''
        hash_string = (self.username + password)
        hash_string = hash_string.encode("utf8")
        return hashlib.sha256(hash_string).hexdigest()
    def check_password(self, password):
        '''Returna True se a senha for válida para
        este usuário, do contrário retorna False'''
        encrypted = self._encrypt_pw(password)
```

```
        return encrypted == self.password

class AuthException(Exception):
    def __init__(self, username, user=None):
        super().__init__(username, user)
        self.username = username
        self.user = user

class UsernameAlreadyExists(AuthException):
    pass

class PasswordTooShort(AuthException):
    pass

class InvalidUsername(AuthException):
    pass

class InvalidPassword(AuthException):
    pass

class Authenticator:
    def __init__(self):
        '''Construtor de um autenticador que gerencia
        logins e logouts de usuários.'''
        self.users = {}

    def add_user(self, username, password):
        if username in self.users:
            raise UsernameAlreadyExists(username)
        if len(password) < 6:
            raise PasswordTooShort(username)
        self.users[username] = User(username, password)

    def login(self, username, password):
        try:
            user = self.users[username]
        except KeyError:
            raise InvalidUsername(username)
        if not user.check_password(password):
            raise InvalidPassword(username, user)
        user.is_logged_in = True
        return True

    def is_logged_in(self, username):
        if username in self.users:
            return self.users[username].is_logged_in
        return False
```

```
authenticator = Authenticator()

class NotLoggedInError(AuthException):
    pass

class NotPermittedError(AuthException):
    pass

class PermissionError(Exception):
    pass

class Authorizor:
    def __init__(self, authenticator):
        self.authenticator = authenticator
        self.permissions = {}

    def add_permission(self, perm_name):
        '''Criar uma nova permissao à qual
        usuários podem estar vinculados'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            self.permissions[perm_name] = set()
        else:
            raise PermissionError("Permissão já existe")

    def permit_user(self, perm_name, username):
        '''Conceder permissão a um usuário'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permissão não existe")
        else:
            if username not in self.authenticator.users:
                raise InvalidUsername(username)
            perm_set.add(username)

    def check_permission(self, perm_name, username):
        if not self.authenticator.is_logged_in(username):
            raise NotLoggedInError(username)
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permissão não existe")
        else:
            if username not in perm_set:
```

```
                raise NotPermittedError(username)
            else:
                return True

authorizor = Authorizor(authenticator)
```

```
Overwriting auth.py
```

[28]:
```
%%file __init__.py
```

```
Writing __init__.py
```

- Agora vamos testar nosso módulo:

[2]:
```
import auth
```

- Primeiro, vamos criar um usário:

[3]:
```
auth.authenticator.add_user("filipe", "filipe_pass")
```

- Em seguida criaremos a ação de leitura:

[4]:
```
auth.authorizor.add_permission("read")
```

- Agora vamos checar se o usuário filipe tem permissão de leitura:

[5]:
```
auth.authorizor.check_permission("read", "filipe")
```

```
---------------------------------------------------------------------------
NotLoggedInError                          Traceback (most recent call last)
<ipython-input-5-509081ad210c> in <module>
----> 1 auth.authorizor.check_permission("read", "filipe")

~/Dropbox/UFRR Docência/POO - Python/poo_python_aulas_2019_2/Cap 04 -␣
↪Tratamento de Exceção/auth.py in check_permission(self, perm_name, username)
    105     def check_permission(self, perm_name, username):
    106         if not self.authenticator.is_logged_in(username):
--> 107             raise NotLoggedInError(username)
    108         try:
    109             perm_set = self.permissions[perm_name]

NotLoggedInError: ('filipe', None)
```

- Como o usuário não está logado, o sistema lança um erro.

[6]:
```
auth.authenticator.is_logged_in("filipe")
```

[6]:
```
False
```

- Vamos fazer o login:

[7]:
```
auth.authenticator.login("filipe", "filipe_pass")
```

[7]:
```
True
```

- Agora vamos verificar novamente se o usuário "filipe" tem permissão de criar leitura:

[8]:
```
auth.authorizor.check_permission("read", "filipe")
```

```
---------------------------------------------------------------------------

NotPermittedError                         Traceback (most recent call last)

<ipython-input-8-509081ad210c> in <module>
----> 1 auth.authorizor.check_permission("read", "filipe")


~/Dropbox/UFRR Docência/POO - Python/poo_python_aulas_2019_2/Cap 04 -␣
↪Tratamento de Exceção/auth.py in check_permission(self, perm_name, username)
    112         else:
    113             if username not in perm_set:
--> 114                 raise NotPermittedError(username)
    115             else:
    116                 return True


NotPermittedError: ('filipe', None)
```

- Observamos que o usuário não tem essa permissão, conforme já esperávemos.
- Vamos agora testar se o esse usuário tem permissão de escrita:

[9]:
```
auth.authorizor.check_permission("write", "filipe")
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)
```

```
        ~/Dropbox/UFRR Docência/POO - Python/poo_python_aulas_2019_2/Cap 04 -␣
↪Tratamento de Exceção/auth.py in check_permission(self, perm_name, username)
        108         try:
    --> 109             perm_set = self.permissions[perm_name]
        110         except KeyError:


        KeyError: 'write'


    During handling of the above exception, another exception occurred:


        PermissionError                           Traceback (most recent call last)

        <ipython-input-9-a70394e06f2d> in <module>
    ----> 1 auth.authorizor.check_permission("write", "filipe")


        ~/Dropbox/UFRR Docência/POO - Python/poo_python_aulas_2019_2/Cap 04 -␣
↪Tratamento de Exceção/auth.py in check_permission(self, perm_name, username)
        109             perm_set = self.permissions[perm_name]
        110         except KeyError:
    --> 111             raise PermissionError("Permissão não existe")
        112         else:
        113             if username not in perm_set:


        PermissionError: Permissão não existe
```

- Note que essa permissão ainda não existe.
- Vamos criá-la e concedê-la ao usuário "filipe":

[10]:
```
auth.authorizor.add_permission("write")
```

[11]:
```
auth.authorizor.permit_user("write", "filipe")
```

- Finalmente, para entendermos melhor nossas exceções e o funcionamento do sistema, vamos criar um simples menu que permite que certos usuários alterem ou testem um programa:

[1]:
```
import auth
# Configurando os usuários de teste e algumas permissões
auth.authenticator.add_user("fulano", "1234567")
auth.authorizor.add_permission("create_user")
auth.authorizor.add_permission("change_program")
auth.authorizor.add_permission("test_program")
auth.authorizor.permit_user("test_program", "fulano")
```

[4]:
```
class Editor:
    def __init__(self):
        self.username = None
        self.menu_map = {
        "login": self.login,
        "test": self.test,
        "change": self.change,
        "create_user": self.create_user,
        "quit": self.quit
        }

    def login(self):
        logged_in = False
        while not logged_in:
            username = input("username: ")
            password = input("password: ")
            try:
                logged_in = auth.authenticator.login(username, password)
                print('Usuário {} logged in!'.format(username))
            except auth.InvalidUsername:
                print("Desculpa, esse usuário nao existe")
            except auth.InvalidPassword:
                print("Desculpe, password incorreto")
            else:
                self.username = username

    def is_permitted(self, permission):
        try:
            auth.authorizor.check_permission(permission, self.username)
        except auth.NotLoggedInError as e:
            print("{} nao está logged in".format(e.username))
            return False
        except auth.NotPermittedError as e:
            print("{} nao pode {}".format(e.username, permission))
            return False
        else:
            return True

    def test(self):
        if self.is_permitted("test_program"):
            print("Testando programa agora...")

    def create_user(self):
        if self.is_permitted("create_user"):
            print("Criando usuário agora...")

    def change(self):
```

```
        if self.is_permitted("change_program"):
            print("Mudando programa agora...")

    def quit(self):
        raise SystemExit()

    def menu(self):
        try:
            answer = ""
            while True:
                print("""
                Please enter a command:
                \tlogin\tLogin
                \ttest\tTest the program
                \tchange\tChange the program
                \tcreate_user\tCreate user
                \tquit\tQuit
                """)
                answer = input("entre com um comando: ").lower()
                try:
                    func = self.menu_map[answer]
                except KeyError:
                    print("{} não é uma opção válida".format(answer))
                else:
                    func()
        finally:
            print("Obrigado por testar o módulo auth")

Editor().menu()
```

```
                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit

entre com um comando: change
None nao está logged in

                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit
```

```
entre com um comando: login
username: fulano
password: 1234567
Usuário fulano logged in!

                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit

entre com um comando: test
Testando programa agora...

                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit

entre com um comando: create_user
fulano nao pode create_user

                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit

entre com um comando: test
Testando programa agora...

                Please enter a command:
                        login   Login
                        test    Test the program
                        change  Change the program
                        create_user     Create user
                        quit    Quit

entre com um comando: quit
Obrigado por testar o módulo auth


        An exception has occurred, use %tb to see the full traceback.
```

```
SystemExit
```