



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

## 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;

## 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;

## 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):

In [1]:

```
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:

In [2]:

```
1 x = 5/0
```

```
-----
-----
ZeroDivisionError                                Traceback (most recent call
last)
<ipython-input-2-fc2abf138dd5> in <module>
----> 1 x = 5/0
```

**ZeroDivisionError:** division by zero

In [9]:

```
1 lista = range(5)
2 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

In [10]:

```
1 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'

In [11]:

```
1 lista.adiciona
```

**AttributeError**

Traceback (most recent call

last)

<ipython-input-11-618bc9227df2> in <module>

----> 1 lista.adiciona

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

In [12]:

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

**KeyError**

Traceback (most recent call

last)

<ipython-input-12-88d0c6fdc283> in <module>

1 d = {'1': 'um'}

----> 2 d['2']

**KeyError:** '2'

In [13]:

```
1 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;

## Lançando exceções

- Podemos usar o mesmo mecanismo 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 usuário tente adicionar um inteiro não par;

In [1]:

```
1 class EvenOnly(list):
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")
7         super().append(valor)
```

In [4]:

```
1 lista = EvenOnly()
2 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 adicio
nados")
      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:

In [5]:

```
1 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 adicio  
nados")  
      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:

In [6]:

```
1 lista.append(2)  
2 lista.append(4)  
3 lista.append(6)  
4 lista
```

Out[6]:

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

## O efeito de uma exceção

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

In [7]:

```
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!"
```

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

In [8]:

```
1 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:

In [9]:

```
1 ▾ def call_excepto():  
2     print("chama uma função que lança exceção...")  
3     no_return()  
4     print("uma exceção foi lançada...")  
5     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\_excepto* nem no *no\_return*:

In [10]:

```
1 call_excepto(r)
```

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\_excepto(r)

<ipython-input-9-853da568a806> in call\_excepto(r)

```
1 def call_excepto(r):  
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

## 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\_return supracitado:

In [11]:

```
1 ▾ try:  
2     no_return()  
3 ▾ except:  
4     print("Exceção capturada")  
5     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 código 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:

In [13]:

```
1 ▾ def funny_division(divider):
2 ▾     try:
3         return 100 / divider
4 ▾     except ZeroDivisionError:
5         return "Zero is not a good idea!"
```

In [14]:

```
1 print(funny_division(0))
```

Zero is not a good idea!

In [15]:

```
1 print(funny_division(50.0))
```

2.0

In [17]:

```
1 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'

In [18]:

```
1 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: 'd
ivider'
```

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



In [8]:

```
1 ▾ def funny_division2(anumber):
2 ▾     try:
3 ▾         if anumber == 13:
4             raise ValueError("13 é um número bloqueado")
5             return 100 / anumber
6 ▾     except ZeroDivisionError:
7             return "Entre com um número diferente de zero"
8 ▾     except TypeError:
9             return "Entre com um valor numérico"
10 ▾    except ValueError:
11        print("13 não!")
12        raise#lança novamente a exceção ValueError
```

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

In [2]:

```
1 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:

In [3]:

```
1 ▾ def funny_division2(anumber):
2 ▾     try:
3 ▾         if anumber == 13:
4             raise ValueError("13 é um número bloqueado")
5             return 100 / anumber
6 ▾     except (ZeroDivisionError, TypeError, ValueError) as e:
7         print("Erro:", e)
```

In [4]:

```
1 funny_division2(13)
```

Erro: 13 é um número bloqueado

In [5]:

```
1 funny_division2("string")
```

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

In [6]:

```
1 funny_division2(0)
```

Erro: division by zero

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

In [7]:

```
1 ▾ try:
2     raise ValueError("Esse é um argumento", 'Outro argumento', 1)
3 ▾ except ValueError as e:
4     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:

In [9]:

```
1 import random
2 def exemplo_excecoes_aleatorias():
3     some_exceptions = [ValueError, TypeError, IndexError, None]
4     try:
5         choice = random.choice(some_exceptions)
6         print("raising {}".format(choice))
7         if choice:
8             raise choice("An error")
9     except ValueError:
10        print("ValueError Capturado")
11    except TypeError:
12        print("TypeError Capturado")
13    except Exception as e:
14        print("Um outro tipo de erro capturado: %s" % (e.__class__.__name__))
15    else:
16        print("Esse código é chamado se não houver nenhuma exceção")
17    finally:
18        print("Esse código é sempre chamado (ainda que haja erro)")
```

In [42]:

```
1 exemplo_excecoes_aleatorias()
```

raising None

Esse código é chamado se não houver nenhuma exceção

Esse código é sempre chamado (ainda que haja erro)

In [43]:

```
1 exemplo_excecoes_aleatorias()
```

raising <class 'TypeError'>

TypeError Capturado

Esse código é sempre chamado (ainda que haja erro)

In [46]:

```
1 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 todas 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](https://docs.python.org/3/tutorial/errors.html) (<https://docs.python.org/3/tutorial/errors.html>):

In [9]:

```
1 ▾ %%file myfile.txt
2   Poo com python é bem legal.
3   Boa vista é uma cidade muito bonita.
4   O Brasil é um lindo país.
```

Writing myfile.txt

In [13]:

```
1 ▾ try:
2     f = open('myfile.txt', 'r')
3 ▾ except OSError:
4     print('Arquivo não pode ser aberto', arg)
5 ▾ else:
6     print('o arquivo tem', len(f.readlines()), 'linhas')
7     f.close()
```

o arquivo 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).

## 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:

In [ ]:

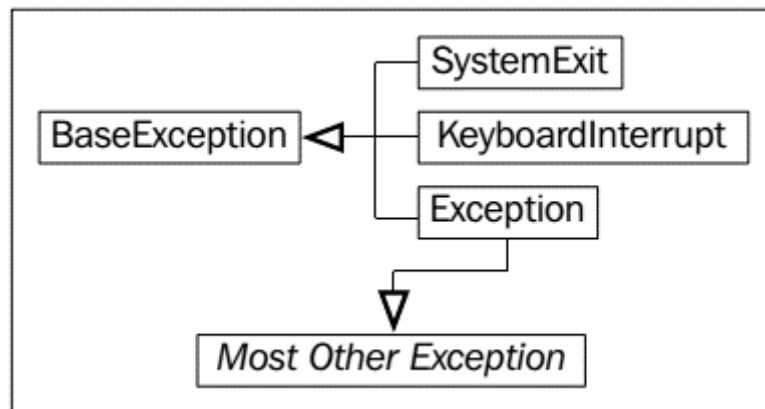
```
1 ▾ with open("myfile.txt") as f:
2 ▾     for line in f:
3         print(line, end="")
```

## 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, tipicamente 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 cláusula except (except Exception);

## 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:

In [2]:

```
1 class InvalidWithdrawal(Exception):
2     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:

In [3]:

```
1 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 fique fácil criar exceções em python, sem precisar sobrescrever o `__init__` da classe `Exception`;
- Entretanto, caso você deseje customizar o `__init__`, você pode:

In [26]:

```
1 class InvalidWithdrawal(Exception):
2     def __init__(self, balance, amount):
3         super().__init__("Nao foi possivel realizar o saque: "
4                           "a conta não tem R${}".format(amount))
5         self.amount = amount
6         self.balance = balance
7     def valor_faltante(self):
8         return self.amount - self.balance
```

In [15]:

```
1 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`:

In [16]:

```
1 try:
2     raise InvalidWithdrawal(25, 50)
3 except InvalidWithdrawal as e:
4     print("O valor que você está tentando sacar "
5           "é maior que o seu saldo. Esta faltando o seguinte valor: "
6           "${}".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:

In [27]:

```
1  ▾ class ATM:
2  ▾     def realizar_operacao(self, operacao):
3  ▾         try:
4  ▾             operacao.realizar()
5  ▾         except InvalidWithdrawal as e:
6  ▾             print("Erro: ", e)
7  ▾         except Exception as e:
8  ▾             print("Erro: ", e)
9
10 ▾ class Conta:
11 ▾     def __init__(self, agencia, numero):
12 ▾         self.agencia = agencia
13 ▾         self.numero = numero
14 ▾         self.saldo = 0.0
15
16 ▾     def sacar(self, valor):
17 ▾         if valor>self.saldo or valor<0.0:
18 ▾             raise InvalidWithdrawal(self.saldo, valor)
19 ▾         self.saldo -= valor
20
21 ▾     def depositar(self, valor):
22 ▾         if valor<0.0:
23 ▾             raise ValueError("Depósito:: Valor Negativo")
24 ▾         self.saldo += valor
25
26 ▾ class Transferencia:
27 ▾     def __init__(self, conta_origem, conta_destino, valor):
28 ▾         self.conta_origem = conta_origem
29 ▾         self.conta_destino = conta_destino
30 ▾         self.valor = valor
31
32 ▾     def realizar(self):
33 ▾         self.conta_origem.sacar(self.valor)
34 ▾         self.conta_destino.depositar(self.valor)
35
36 ▾ class Deposito:
37 ▾     def __init__(self, conta_destino, valor):
38 ▾         self.conta_destino = conta_destino
39 ▾         self.valor = valor
40
41 ▾     def realizar(self):
42 ▾         self.conta_destino.depositar(self.valor)
43
44 ▾ class Saque:
45 ▾     def __init__(self, conta, valor):
46 ▾         self.conta = conta
47 ▾         self.valor = valor
48
49 ▾     def realizar(self):
50 ▾         self.conta.sacar(self.valor)
```

In [28]:

```
1  atm = ATM()
```

In [29]:

```
1 c1 = Conta("2121-3", "31314-0")
2 c1.depositar(100)
3 c2 = Conta("2121-3", "31314-0")
4 c2.depositar(200)
```

In [30]:

```
1 operacao_saque = Saque(c1, 150)
2 atm.realizar_operacao(operacao_saque)
```

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

## 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;
- Pensando 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 autenticação (*Authenticator*), que gerenciar os logins e logouts;
- Teremos ainda a classe *Authorizer* que será responsável por gerenciar as permissões dos usuários 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*:



In [21]:

```
1 import hashlib
2
3 class User:
4     def __init__(self, username, password):
5         '''Cria um novo usuário. A senha é
6         criptografada depois que é salva.'''
7         self.username = username
8         self.password = self._encrypt_pw(password)
9         self.is_logged_in = False
10    def _encrypt_pw(self, password):
11        '''Criptografa o password e depois retorna o sha.'''
12        hash_string = (self.username + password)
13        hash_string = hash_string.encode("utf8")
14        return hashlib.sha256(hash_string).hexdigest()
15    def check_password(self, password):
16        '''Retorna True se a senha for válida para
17        este usuário, do contrário retorna False'''
18        encrypted = self._encrypt_pw(password)
19        return encrypted == self.password
```

- Como de praxe, vamos testar nossa classe User:

In [22]:

```
1 u = User('filipe', '123456')
```

In [23]:

```
1 u.check_password('123456')
```

Out[23]:

True

In [24]:

```
1 u.check_password('12345')
```

Out[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:

In [25]:

```
1 ▾ class AuthException(Exception):
2 ▾     def __init__(self, username, user=None):
3         super().__init__(username, user)
4         self.username = username
5         self.user = user
6
7 ▾ class UsernameAlreadyExists(AuthException):
8     pass
9
10 ▾ class PasswordTooShort(AuthException):
11     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:

In [39]:

```
1 ▾ class Authenticator:
2 ▾     def __init__(self):
3         '''Construtor de um autenticador que gerencia
4         logins e logouts de usuários.'''
5         self.users = {}
6 ▾     def add_user(self, username, password):
7 ▾         if username in self.users:
8             raise UsernameAlreadyExists(username)
9 ▾         if len(password) < 6:
10             raise PasswordTooShort(username)
11         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:

In [40]:

```
1 ▾ class InvalidUsername(AuthException):
2     pass
3 ▾ class InvalidPassword(AuthException):
4     pass
```

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

In [ ]:

```
1 ▾ def login(self, username, password):
2 ▾     try:
3         user = self.users[username]
4 ▾     except KeyError:
5         raise InvalidUsername(username)
6 ▾     if not user.check_password(password):
7         raise InvalidPassword(username, user)
8     user.is_logged_in = True
9     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*:

In [41]:

```
1 ▾ def is_logged_in(self, username):
2 ▾     if username in self.users:
3         return self.users[username].is_logged_in
4     return False
```

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

In [ ]:

```
1 authenticator = Authenticator()
```

- Agora podemos criar nossa classe `Authorizer`;
- 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:

In [26]:

```
1  class NotLoggedInError(AuthException):
2      pass
3
4  class NotPermittedError(AuthException):
5      pass
6
7  class PermissionError(Exception):
8      pass
9
10 class Authorizer:
11     def __init__(self, authenticator):
12         self.authenticator = authenticator
13         self.permissions = {}
14
15     def add_permission(self, perm_name):
16         '''Crie uma nova permissão à qual os usuários
17         possam ser adicionados'''
18         try:
19             perm_set = self.permissions[perm_name]
20         except KeyError:
21             self.permissions[perm_name] = set()
22         else:
23             raise PermissionError("Essa permissao ja Existe")
24
25     def permit_user(self, perm_name, username):
26         '''Concede permissão ao usuário'''
27         try:
28             perm_set = self.permissions[perm_name]
29         except KeyError:
30             raise PermissionError("Sem permissão")#ATUALIZAR str
31         else:
32             if username not in self.authenticator.users:
33                 raise InvalidUsername(username)
34             perm_set.add(username)
```

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

In [ ]:

```
1  authorizer = Authorizer(authenticator)
```

- Abaixo vamos colocar tudo junto no nosso módulo auth:

In [1]:

```
1 ▾ %%file auth.py
2
3 import hashlib
4
5 ▾ class User:
6 ▾     def __init__(self, username, password):
7         '''Cria um novo usuário. A senha é
8         criptografada depois que é salva.'''
9         self.username = username
10        self.password = self._encrypt_pw(password)
11        self.is_logged_in = False
12 ▾     def _encrypt_pw(self, password):
13         '''Criptografa o password e depois retorna o sha.'''
14         hash_string = (self.username + password)
15         hash_string = hash_string.encode("utf8")
16         return hashlib.sha256(hash_string).hexdigest()
17 ▾     def check_password(self, password):
18         '''Retorna True se a senha for válida para
19         este usuário, do contrário retorna False'''
20         encrypted = self._encrypt_pw(password)
21         return encrypted == self.password
22
23 ▾ class AuthException(Exception):
24 ▾     def __init__(self, username, user=None):
25         super().__init__(username, user)
26         self.username = username
27         self.user = user
28
29 ▾ class UsernameAlreadyExists(AuthException):
30     pass
31
32 ▾ class PasswordTooShort(AuthException):
33     pass
34
35 ▾ class InvalidUsername(AuthException):
36     pass
37
38 ▾ class InvalidPassword(AuthException):
39     pass
40
41 ▾ class Authenticator:
42 ▾     def __init__(self):
43         '''Construtor de um autenticador que gerencia
44         logins e logouts de usuários.'''
45         self.users = {}
46
47 ▾     def add_user(self, username, password):
48 ▾         if username in self.users:
49             raise UsernameAlreadyExists(username)
50 ▾         if len(password) < 6:
51             raise PasswordTooShort(username)
52         self.users[username] = User(username, password)
53
54 ▾     def login(self, username, password):
55 ▾         try:
56             user = self.users[username]
57 ▾         except KeyError:
58             raise InvalidUsername(username)
59 ▾         if not user.check_password(password):
```

```

60         raise InvalidPassword(username, user)
61     user.is_logged_in = True
62     return True
63
64     def is_logged_in(self, username):
65         if username in self.users:
66             return self.users[username].is_logged_in
67         return False
68
69     authenticator = Authenticator()
70
71     class NotLoggedInError(AuthException):
72         pass
73
74     class NotPermittedError(AuthException):
75         pass
76
77     class PermissionError(Exception):
78         pass
79
80     class Authorizer:
81         def __init__(self, authenticator):
82             self.authenticator = authenticator
83             self.permissions = {}
84
85         def add_permission(self, perm_name):
86             '''Criar uma nova permissao à qual
87             usuários podem estar vinculados'''
88             try:
89                 perm_set = self.permissions[perm_name]
90             except KeyError:
91                 self.permissions[perm_name] = set()
92             else:
93                 raise PermissionError("Permissão já existe")
94
95         def permit_user(self, perm_name, username):
96             '''Conceder permissão a um usuário'''
97             try:
98                 perm_set = self.permissions[perm_name]
99             except KeyError:
100                 raise PermissionError("Permissão não existe")
101             else:
102                 if username not in self.authenticator.users:
103                     raise InvalidUsername(username)
104                 perm_set.add(username)
105
106         def check_permission(self, perm_name, username):
107             if not self.authenticator.is_logged_in(username):
108                 raise NotLoggedInError(username)
109             try:
110                 perm_set = self.permissions[perm_name]
111             except KeyError:
112                 raise PermissionError("Permissão não existe")
113             else:
114                 if username not in perm_set:
115                     raise NotPermittedError(username)
116                 else:
117                     return True
118
119     authorizer = Authorizer(authenticator)

```

## Overwriting auth.py

In [28]:

```
1 ▾ %%file __init__.py
2
```

Writing \_\_init\_\_.py

- Agora vamos testar nosso módulo:

In [2]:

```
1 import auth
```

- Primeiro, vamos criar um usuário:

In [3]:

```
1 auth.authenticator.add_user("filipe", "filipe_pass")
```

- Em seguida criaremos a ação de leitura:

In [4]:

```
1 auth.authorizer.add_permission("read")
```

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

In [5]:

```
1 auth.authorizer.check_permission("read", "filipe")
```

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

~/Dropbox/UFRER Docência/P00 - Python/poo_python_aulas_2019_2/Cap 04 -
Tratamento de Exceção/auth.py in check_permission(self, perm_name, us
ername)
    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.

In [6]:

```
1 auth.authenticator.is_logged_in("filipe")
```

Out[6]:

False

- Vamos fazer o login:

In [7]:

```
1 auth.authenticator.login("filipe", "filipe_pass")
```

Out[7]:

True

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

In [8]:

```
1 auth.authorizer.check_permission("read", "filipe")
```

```
-----  
-----  
NotPermittedError                                Traceback (most recent call  
last)  
<ipython-input-8-509081ad210c> in <module>  
----> 1 auth.authorizer.check_permission("read", "filipe")  
  
~/Dropbox/UFRR Docência/P00 - Python/poo_python_aulas_2019_2/Cap 04 -  
Tratamento de Exceção/auth.py in check_permission(self, perm_name, us  
ername)  
    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 esperávamos.
- Vamos agora testar se o esse usuário tem permissão de escrita:



In [9]:

```
1 auth.authorizor.check_permission("write", "filipe")
```

```
-----  
-----  
KeyError                                Traceback (most recent call  
last)  
~/Dropbox/UFRR Docência/P00 - Python/poo_python_aulas_2019_2/Cap 04 -  
Tratamento de Exceção/auth.py in check_permission(self, perm_name, us  
ername)  
    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/P00 - Python/poo_python_aulas_2019_2/Cap 04 -  
Tratamento de Exceção/auth.py in check_permission(self, perm_name, us  
ername)  
    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":

In [10]:

```
1 auth.authorizor.add_permission("write")
```

In [11]:

```
1 auth.authorizor.permit_user("write", "filipe")
```

- Finalmente, para entendermos melhor nossas exceções e o funcionamento do sistema, vamos criar um simples menu que permite um usuário alterar ou testar um programa:

In [1]:

```
1  import auth
2  # Configurando os usuários de teste e algumas permissões
3  auth.authenticator.add_user("fulano", "1234567")
4  auth.authorizer.add_permission("create_user")
5  auth.authorizer.add_permission("change_program")
6  auth.authorizer.add_permission("test_program")
7  auth.authorizer.permit_user("test_program", "fulano")
```

In [4]:

```
1  ▾ class Editor:
2  ▾     def __init__(self):
3         self.username = None
4         self.menu_map = {
5             "login": self.login,
6             "test": self.test,
7             "change": self.change,
8             "create_user": self.create_user,
9             "quit": self.quit
10        }
11
12  ▾     def login(self):
13         logged_in = False
14         while not logged_in:
15             username = input("username: ")
16             password = input("password: ")
17             try:
18                 logged_in = auth.authenticator.login(username, password)
19                 print('Usuário {} logged in!'.format(username))
20             except auth.InvalidUsername:
21                 print("Desculpa, esse usuário nao existe")
22             except auth.InvalidPassword:
23                 print("Desculpe, password incorreto")
24             else:
25                 self.username = username
26
27  ▾     def is_permitted(self, permission):
28             try:
29                 auth.authorizer.check_permission(permission, self.username)
30             except auth.NotLoggedInError as e:
31                 print("{} nao está logged in".format(e.username))
32                 return False
33             except auth.NotPermittedError as e:
34                 print("{} nao pode {}".format(e.username, permission))
35                 return False
36             else:
37                 return True
38
39  ▾     def test(self):
40         if self.is_permitted("test_program"):
41             print("Testando programa agora...")
42
43  ▾     def create_user(self):
44         if self.is_permitted("create_user"):
45             print("Criando usuário agora...")
46
47  ▾     def change(self):
48         if self.is_permitted("change_program"):
49             print("Mudando programa agora...")
50
51  ▾     def quit(self):
52         raise SystemExit()
53
54  ▾     def menu(self):
55         try:
56             answer = ""
57             while True:
58                 print("""
59                 Please enter a command:
```

```

60         \tlogin\tLogin
61         \ttest\tTest the program
62         \tchange\tChange the program
63         \tcreate_user\tCreate user
64         \tquit\tQuit
65         """)
66         answer = input("entre com um comando: ").lower()
67     try:
68         func = self.menu_map[answer]
69     except KeyError:
70         print("{} não é uma opção válida".format(answer))
71     else:
72         func()
73 finally:
74     print("Obrigado por testar o módulo auth")
75
76 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