Tratamento de Exceções

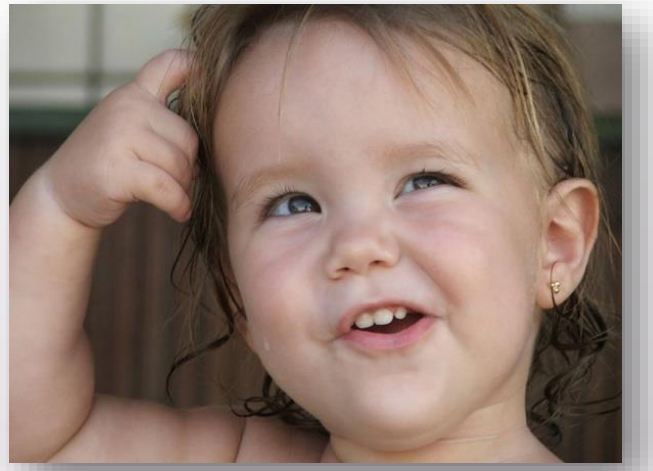
Prof. Fábio Procópio

Prof. João Nascimento

Técnico Integrado em Informática – Programação Estruturada e Orientada a Objetos

Relembrando...

- Na aula passada, discutimos sobre classes abstratas;
- Nesta aula, vamos estudar como tratar erros que são gerados por exceções do Python.



Introdução

- Uma instrução de um programa, mesmo sintaticamente correta, ela pode ocasionar um erro durante sua execução;

- Os **erros detectados** durante a execução de um programa são chamados de **exceções**;
  - Uma exceção não indica, obrigatoriamente, que um erro é fatal
  - As exceções são usadas para indicar que algo anormal aconteceu

- Muitas vezes, os programas não tratam exceções e, portanto, resultam em mensagens de erro.

```
>>> print(res)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(res)
NameError: name 'res' is not defined
```

Classe Exception

- **Exception** é a **super-classe** da qual todas as outras **exceções** são descendentes;
- Pode ser usada para tratar qualquer erro que possa surgir na execução de uma instrução;
- Vamos criar duas situações: a) sem tratamento de exceções e b) com tratamento de exceções. Teste os dois cenários informando o **valor 0 para a variável Y**. Observe o que ocorrerá.

```
/*Situação a*/

x = int(input("Digite um valor para x: "))
y = int(input("Digite um valor para y: "))
print(x / y)
```

```
/*Situação b*/

try:
  x = int(input("Digite um valor para x: "))
  y = int(input("Digite um valor para y: "))
  print(x / y)
except Exception:
  print("ERRO: Algo não funcionou bem.")
```

Algumas subclasses de Exception

Classe	Descrição
SintaxError	erro de sintaxe no código
NameError	nome (variável) não encontrado
AttributeError	falha na atribuição de valores
TypeError	operação com tipos incompatíveis
ValueError	tipo da operação é correta, mas o valor é ilegal
IOError	erro na tentativa de leitura/escrita
ZeroDivisionError	tentativa de divisão por zero
IndexError	acesso a um índice não existente
KeyError	acesso a uma chave não existente

Escolha do tipo de exceção

- Para que mensagens de erro sejam adequadas para cada tipo de erro, é necessário usar a exceção específica;

- Por exemplo, a exceção disparada quando ocorre uma tentativa de divisão por zero é a **ZeroDivisionError**
  - Para isso, informe 0 para o valor de `y`

```
try:
  x = int(input("Digite um valor para x: "))
  y = int(input("Digite um valor para y: "))
  print(x / y)
except ZeroDivisionError:
  print("ERRO: Impossível efetuar divisão por 0.")
```

Exceção não identificada

- Nem sempre uma **exceção específica** é programada para ser disparada. Como assim?
  - O programador pode não listar todas as exceções que possam surgir no trecho em que ele "protegeu" contra erros
  - Para casos como esse, podemos deixar o tratamento com a super-classe Exception
  - Caso a(s) exceção(ões) específica(s) não identifique(m) o erro, a super-classe o fará

- Para simular essa situação, digite a *string* `Olá` para o valor de x

```
try:
  x = int(input("Digite um valor para x: "))
  y = int(input("Digite um valor para y: "))
  print(x / y)
except ZeroDivisionError:
  print("ERRO: Impossível efetuar divisão por 0.")
except Exception as erro:
  print("Erro: {}".format(erro))
```

else – 1 de 2

- **ATENÇÃO:** este **`else`** não é o mesmo da estrutura condicional if;
- Os comandos subordinados ao **`else`** **são executados** quando o bloco de comandos subordinado ao **`try`** ocorreu com **sucesso**;
- Em outras palavras, a lógica seria a seguinte:

```
Se ocorreu erro dentro do try: executar comandos dentro do except
Mas, se nada de errado ocorreu: executar comandos dentro do else
```

else – 2 de 2

```
try:
  x = int(input("Digite um valor para x: "))
  y = int(input("Digite um valor para y: "))
  res = x / y
except ZeroDivisionError:
  print("ERRO: Impossível efetuar divisão por 0.")
except Exception as erro:
  print("Erro: {}".format(erro))
else:
  print("Resultado da divisão: {:.2f}".format(res))
```

finally – 1 de 2

- Os comandos subordinados ao **`finally` são executados independentemente de** ocorrer (ou não) **erro**
  - Ou seja, sempre será executado

- Em outras palavras, a lógica seria a seguinte:

```
Se ocorreu erro dentro do try: executar comandos dentro do except
Mas, se nada de errado ocorreu: executar comandos dentro do else
Comandos do bloco finally sempre serão executados
```

finally – 2 de 2

```
try:
  x = int(input("Digite um valor para x: "))
  y = int(input("Digite um valor para y: "))
  res = x / y
except ZeroDivisionError:
  print("ERRO: Impossível efetuar divisão por 0.")
except Exception as erro:
  print("Erro: {}".format(erro))
else:
  print("Resultado da divisão: {:.2f}".format(res))
finally:
  aval = int(input("Gostou do programinha?\ Dê sua nota de 1 a 5: ")
  print("Obrigado pela sua avaliação: {}".format(aval))
```

Tratamento de Erros

- Então o tratamento de erros, em Python, é dado como se vê abaixo:

```
try:
  # Código a ser executado "sob vigilância"
except Exception:
  # Caso haja falha no bloco anterior, trate-a aqui
else:
  # Se o bloco try não produziu erro, execute este bloco
finally:
  # Sempre será executado, independente de erros
```

Aplicação de exceções – 1 de 3

```
class Professor:
  def __init__(self, matricula = None, nome = None):
    self.matricula = matricula
    self.nome = nome
def cadastra(self):
    nome_arquivo = "professor.txt"
    try:
      arquivo = open(nome_arquivo, "r")
      linha = arquivo.readlines()
      linha.append(str(self.matricula) + "|" + self.nome + "\n")
      arquivo = open(nome_arquivo, "w")
      arquivo.writelines(linha)
    except IOError:
      print("ERRO: Arquivo não localizado.")
    except Exception as erro:
      print("ERRO: {}".format(erro))
    else:
      print("Dados gravados com sucesso.")
    finally:
      arquivo.close()
```

Aplicação de exceções – 2 de 3

```
# Continuação da classe Professor

def lista(self):
    nome_arquivo = "professor.txt"
    try:
      arquivo = open(nome_arquivo, "r")
      for linha in arquivo.readlines():
        campos = linha.strip()
        matricula, nome = campos.split("|")
        print("Matrícula: {}".format(matricula))
        print("Nome.....: {}\n".format(nome))
    except IOError:
      print("ERRO: Arquivo não localizado.")
    except Exception as erro:
      print("ERRO: {}".format(erro))
    else:
      print("Dados gravados com sucesso.")
    finally:
      arquivo.close()
```

Aplicação de exceções – 3 de 3

```
from Professor import *

prof = Professor(11, "Fábio Procópio")
prof.cadastra()
prof = Professor(22, "Rodrigo Siqueira")
prof.cadastra()
prof = Professor(33, "João Nascimento")
prof.cadastra()

prof.lista()
```

Canal no Youtube

Em nosso canal no Youtube, você pode acessar uma playlist que preparamos com várias vídeo-aulas falando sobre o assunto. Veja:

- Tratamento de Exceções

Referências

1. THE PYTHON TUTORIAL. **Errors and Exception**. Disponível em: https://docs.python.org/2.7/tutorial/errors.html. Acessado em: 28 out. 2019.

2. PYTHON REFERENCE (THE RIGHT WAY). **strip**. Disponível em: http://python-reference.readthedocs.io/en/latest/docs/str/strip.html. Acessado em: 28 out. 2019.