VOLTAR

Tratamento de exceções

Apresentar os comandos que realizam o tratamento de exceções ou erros quando estes ocorrem no código.

NESTE TÓPICO

Marcar tópico

Olá alunos,

Já percebemos que quando cometemos um erro criando o código e passamos por um compilador ou interpretador, este devolve uma mensagem de erro.

Como professor, já percebi que alguns alunos nem olham a mensagem de erro devolvida pelo interpretador e fecham a mensagem, mas para nós, da área de TI, necessitamos aprender a analisar as mensagens do interpretador ou do compilador para descobrir qual é o erro que estamos cometendo.

Às vezes, o interpretador mostra o número da linha onde ocorreu o erro, mas este pode estar nas linhas acima ou abaixo no código. Isto ocorre porque o interpretador obedece a uma hierarquia do código, por exemplo, se você está codificando com orientação a objeto, criou uma classe e o erro ocorreu em uma das linhas dentro dessa classe, o interpretador vai apontar o erro na linha da definição da classe.

É normal cometermos erros enquanto estamos codificando, mas estes erros ocorrem em tempo de compilação. Agora, temos que ter consciência que os erros podem ocorrer em tempo de execução, ou seja, quando o usuário estiver utilizando o sistema.

É dever também, dos desenvolvedores, prever que o usuário pode cometer erros no momento em que estiverem utilizando o programa e tratar estes erros. Este tipo de ação é chamada de tratamento de exceções (não é politicamente correto dizer que o usuário pode cometer erros e sim exceções! :)).

TRATAMENTO DE EXCEÇÕES (ERROS)

Na maioria dos casos, o tratamento de exceções, é o retorno de uma mensagem ao usuário, porém, se não fizermos isto e ocorrer algum imprevisto, o interpretador devolverá a mensagem padrão dele e o pior que no idioma inglês, o que poderá deixar o usuário perdido e neste caso, poderemos retornar uma mensagem mais amigável e esclarecedora ao usuário.

Na linguagem Python, como em outras linguagens, temos o recurso de tratar exceções.

Então, vamos ver como fazemos isto em Python.

Em Java, temos que marcar o bloco onde pode ocorrer o erro com o comando try e tratar o erro com o comando catch, em PL utilizamos o comando exception. Em Python, marcamos o bloco onde pode ocorrer o erro com **try** e tratamos o erro na seção **except**:

```
try:
    comandos
except nome_exceção:
    comandos
```

O comando **try** (note que vem seguido de dois pontos **":"**) monitora o bloco de comandos que estiverem logo após os dois pontos e se ocorrer algum erro em alguma destas linhas deste bloco, o interpretador irá para a seção **except** e se o erro foi tratado nesta seção, será executado o comando correspondente. Temos que declarar o nome da exceção que na realidade é uma classe da biblioteca Python.

Vamos ver o seguinte exemplo:

```
1.  # divisão por dois números
2.  numero1 = int(input('Entre com o primeiro número: '))
3.  numero2 = int(input('Entre com o segundo número: '))
4.  resultado = numero1 / numero2
5.  print("O resultado é: ",resultado)
6.  input('Pressione ENTER para sair...')
```

Neste exemplo, estamos entrando com dois números (dividendo e divisor) para serem divididos. Após a entrada dos números, dividimos um pelo outro e o resultado é atribuído na variável **resultado**. Em seguida, o resultado é mostrado quando for executada a linha 5.

Agora, vamos supor que o usuário entre com um valor para o primeiro número: 12 e depois, para o segundo número: 0 (zero), como não existe divisão por zero, isto provocará uma exceção, lembro que na calculadora de bolso antiga, quando ocorria isto, o visor da calculadora mostrava "E" e travava (tínhamos que desligar e ligar novamente). No caso do exemplo, o interpretador mostrou a seguinte mensagem:

```
Entre com o primeiro número: 12
Entre com o segundo número: 0
Traceback (most recent call last):
  File "C:/Users/djsch/OneDrive/Área de Trabalho/divisao.py", line 4, in <module>
    resultado = numero1 / numero2
ZeroDivisionError: division by zero
```

Então vamos agora criar um tratamento de erro para o mesmo código:

```
# divisão por dois números
numero1 = int(input('Entre com o primeiro número: '))
numero2 = int(input('Entre com o segundo número: '))
try:
    resultado = numero1 / numero2
    print("O resultado é: ",resultado)
except ZeroDivisionError:
    print("Não é possível divisão por zero")
input('Pressione ENTER para sair...')
```

Na linha **4**, colocamos a marcação com o **try** (seguido de dois pontos). As próximas linhas **5** e **6** serão monitoradas pelo **try**, se alguma exceção ocorrer neste bloco, a seção **except** será localizada e executado o comando da linha **8**.

Na linha **7**, abrimos a seção **except** para tratar o erro, passando o nome deste erro ou exceção: **ZeroDivisionError**. Note que utilizamos o nome do erro do Python, exibido na mensagem de erro do interpretador acima.

Em resumo, se ocorrer uma divisão por zero, e como esta execução está dentro do **try**, e este erro está determinado na seção **except**, a mensagem da linha **8** será exibida.

PRESTE ATENÇÃO:

Na indentação: as linhas **5** e **6** estão com recuo porque estão no **try**. A linha **8** está com recuo porque pertence a seção **except**. A linha **9** **não** pertence ao bloco de tratamento de exceção.

Agora veja o resultado:

```
Entre com o primeiro número: 12
Entre com o segundo número: 0
Não é possível divisão por zero
Pressione ENTER para sair...
```

Mas, se executarmos o mesmo código, sem ocasionar erros, a linha **6** será executada normalmente:

```
Entre com o primeiro número: 12
Entre com o segundo número: 2
O resultado é:  6.0
Pressione ENTER para sair...
```

Agora, vamos supor outros tipos de erros, no mesmo exemplo:

```
Entre com o primeiro número: a
Traceback (most recent call last):
  File "C:/Users/djsch/OneDrive/Área de Trabalho/divisao.py", line 2, in <module>
    numero1 = int(input('Entre com o primeiro número: '))
ValueError: invalid literal for int() with base 10: 'a'
```

```
Entre com o primeiro número: 5,5
Traceback (most recent call last):
  File "C:/Users/djsch/OneDrive/Área de Trabalho/divisao.py", line 2, in <module>
    numero1 = int(input('Entre com o primeiro número: '))
ValueError: invalid literal for int() with base 10: '5,5'
```

O usuário, no primeiro caso, entrou com um valor de string (letra **a**) em uma variável tipo numérica ou poderia ter, acidentalmente, pressionado ENTER, que ocorreria este erro.

No segundo caso, o usuário entrou com um valor **5,5** em vez da notação inglesa: **5.5**, lembrando que na notação inglesa, a vírgula é só um separador de milhares enquanto o ponto, representa a parte fracionária. Exemplo: US$ 1,245,796.47 (um milhão, duzentos e quarenta e cinco mil, setecentos e noventa e seis dólares e quarenta e sete centavos).

Podemos também tratar este erro no mesmo código:

```
# divisão por dois números
try:
    numero1 = float(input('Entre com o primeiro número: '))
    numero2 = float(input('Entre com o segundo número: '))
    numero1 = numero1 / numero2
    print("O resultado é: ",numero1)
except ValueError:
    print("Valor inválido!")
except ZeroDivisionError:
    print("Não é possível divisão por zero")
input('Pressione ENTER para sair...')
```

Colocamos a marcação do bloco **try** no início do programa, senão de nada adiantará deixarmos a marcação na posição em que estava antes. Pois o erro ocorreria da mesma forma, porque o erro estaria fora do bloco **try**. Mudamos o tipo da variável **numero1** e **numero2** para **float**.

Abrimos duas seções **except** e utilizamos a mesma mensagem de erro do interpretador (**ValueError**) . Assim, se o usuário digitar letras, espaço em branco, valores com vírgulas, etc, os erros serão tratados, devolvendo mensagens ao usuário.

Vejam nos exemplos abaixo, o resultado na execução do código quando entramos com valores errados:

```
1.  Entre com o primeiro número: 4
2.  Entre com o segundo número: 0
3.  Não é possível divisão por zero
4.  Pressione ENTER para sair...
```

```
1.  Entre com o primeiro número: 4
2.  Entre com o segundo número: hh
3.  Valor inválido!
4.  Pressione ENTER para sair...
```

```
1.  Entre com o primeiro número: 4,4
2.  Valor inválido!
3.  Pressione ENTER para sair...
```

```
1.  Entre com o primeiro número: 4.4
2.  Entre com o segundo número: 2
3.  O resultado é:  2.2
4.  Pressione ENTER para sair...
```

SAIBA MAIS...

Dê uma olhada nos links abaixo para saber mais sobre a linguagem Python:

https://www.python.org/doc/ (https://www.python.org/doc/)

https://wiki.python.org/moin/PythonBooks (https://wiki.python.org/moin/PythonBooks)

Neste tópico vimos como podemos marcar com a cláusula try um bloco de comandos onde poderá ocorrer algum tipo de erro ou exceção e tratarmos este erro na seção except, quando o código for executado pelo usuário.

Quiz

Exercício Final

Tratamento de exceções

INICIAR

Referências

SUMMERFIELD, M. *Programação em Python 3*: Uma introdução completa à linguagem Python. Rio de Janeiro Alta Books, 2012. 495 p.

MENEZES, N. N. C. *Introdução à programação com Python:* algoritmos e lógica de programação para iniciantes. 2. ed. São Paulo: Novatec, 2014. 328 p.

SWEIGART, AL. *Automatize tarefas maçantes com Python:* programação prática para verdadeiros iniciantes. São Paulo: Novatec, 2015. 568 p.

PYTHON, doc. Disponível em: <https://www.python.org/doc/>. Acesso em: Junho/2018.

PYTHON, books. Disponível em: <https://wiki.python.org/moin/PythonBooks>. Acesso em: Junho/2018.

Avalie este tópico

ANTERIOR
Estruturas de repetição: while e for

Índice

PRÓXIMO
Expressões regulares em Python