Exceções

Exceções são uma forma de captar erros e tratá-los da maneira que você achar melhor em seu código, ao invés de simplesmente retornar uma mensagem de erro padrão do Python.

O try

O tratamento de exceções no Python é feito através de blocos "*try / except*". A ideia já é dada pelo próprio nome. O Python vai "tentar" executar o bloco de código indentado sob a declaração "*try*". Se ocorrer algum erro, você deve definir blocos "*except*", de exceção, que serão rodados no caso de diferentes tipos de erro. Vejamos nosso primeiro exemplo, sem qualquer tratamento de exceções. Vamos fazer com que o Python retorne um erro, inserindo um segundo número igual a 0. Isso resultará numa divisão por 0, o que não gera um erro no Python:

```
print("Vamos dividir dois números inseridos por você\n")
num1 = input("Insira o primeiro número: ")
num2 = input("Insira o segundo número: ")

resultado = int(num1) / int(num2)
print("O resultado é " + str(resultado))

> Vamos dividir dois números inseridos por você

> Insira o primeiro número: 12
> Insira o segundo número: 0
> Traceback (most recent call last):
>  File "exemplos_excecoes.py", line 5, in <module>
>    resultado = int(num1) / int(num2)
> ZeroDivisionError: division by zero
```

Bem, temos esta mensagem de erro do Python, que até é bem clara com relação ao erro que temos. Mas se queremos tratar este erro com mais elegância, vamos incrementar nosso código com blocos "*try / except*":

```
print("Vamos dividir dois números inseridos por você\n")
num1 = input("Insira o primeiro número: ")
num2 = input("Insira o segundo número: ")

try:
  resultado = int(num1) / int(num2)
  print("O resultado é " + str(resultado))
except ZeroDivisionError:
  print("O segundo número não pode ser zero")

> Vamos dividir dois números inseridos por você
```

```
> Insira o primeiro número: 12
> Insira o segundo número: 0
> O segundo número não pode ser zero
```

Repare que agora recebemos a mensagem de erro que definimos, sem aquelas informações adicionais do Python. Se colocarmos números corretamente, o resultado sai certo também, pois nenhum erro é detectado e o bloco de código dentro do "try" é executado até o fim:

```
> Vamos dividir dois números inseridos por você

> Insira o primeiro número: 12
> Insira o segundo número: 4
> O resultado é 3.0
```

Podemos ter mais de uma exceção. No nosso exemplo, por exemplo, temos que pensar também que o usuário pode colocar um valor diferente de um número. Qual o erro teremos neste caso?

```
> Vamos dividir dois números inseridos por você

> Insira o primeiro número: felipe
> Insira o segundo número: 12
> Traceback (most recent call last):
>  File "exemplos_excecoes.py", line 6, in <module>
>    resultado = int(num1) / int(num2)
> ValueError: invalid literal for int() with base 10: 'felipe'
```

Como podemos ver, é um erro do tipo "ValueError". Vamos incluir esta exceção em nosso código também. Ela virá logo após da primeira exceção que já definimos.

```
try:
  resultado = int(num1) / int(num2)
  print("O resultado é " + str(resultado))
except ZeroDivisionError:
  print("O segundo número não pode ser zero")
except ValueError:
  print("Você deve inserir dois números")
```

Agora, vamos fazer o teste desta nova exceção:

```
> Vamos dividir dois números inseridos por você

> Insira o primeiro número: felipe
> Insira o segundo número: 12
> Você deve inserir dois números
```

Funcionando perfeitamente. Você pode utilizar quantas exceções quiser, pegando todos os tipos de erro que podem ser cometidos pelo usuário em cada caso. E se eu

quiser pegar todos os erros? Basta usar somente o except, sem qualquer definição do tipo de erro. Mas algumas literaturas não consideram isto como uma boa prática, pois ele não identifica a raiz do problema e também não permite passar uma informação mais completa e precisa para o usuário. Mas de qualquer forma, ficaria assim:

```
try:
  resultado = int(num1) / int(num2)
  print("O resultado é " + str(resultado))
except:
  print("Uma das entradas é inválida. Favor inserir dois números, sendo o segundo
diferente que zero")

> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: 12
> Insira o segundo número: 0
> Uma das entradas é inválida. Favor inserir dois números, sendo o segundo diferente que
> zero

> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: felipe
> Insira o segundo número: 1
> Uma das entradas é inválida. Favor inserir dois números, sendo o segundo diferente que
> zero
```

No exemplo acima, rodamos o código duas vezes, uma para cada erro que já vimos. O tratamento para ambos é o mesmo, com a mesma mensagem exibida ao usuário.

Vou manter o código anterior, com o tratamento das exceções pelo tipo de erro, e comentar o código acima para prosseguir.

Para ver todos os tipos de exceção, acesse o link abaixo:
https://docs.python.org/3/library/exceptions.html

else e finally

Temos mais duas estruturas que podemos usar com as exceções no Python, que são o “else” e o “finally”. O “else” é um bloco final, que roda após o código no bloco “try” rodar sem qualquer erro. Vejamos:

```
try:
  resultado = int(num1) / int(num2)
  print("O resultado é " + str(resultado))
except ZeroDivisionError:
  print("O segundo número não pode ser zero")
except ValueError:
  print("Você deve inserir dois números")
else:
  print("Divisão feita!")
```

Utilizando os números adequados, veremos a mensagem final ser impressa, conforme está definido no bloco “else”. Caso caia na exceção, o “else” não é executado:

```
> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: 12
> Insira o segundo número: 3
> O resultado é 4.0
> Divisão feita!

> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: 12
> Insira o segundo número: 0
> O segundo número não pode ser zero
```

Repare que o código neste bloco “*else*” só é executado quando não incorremos em qualquer exceção, ou seja, o bloco “*try*” é executado até o final sem qualquer tipo de erro.

Já o “*finally*” define um bloco que sempre será executado, independente de ocorrer um erro ou do bloco definido no “*try*” ter sido executado sem erros.

```
try:
  resultado = int(num1) / int(num2)
  print("O resultado é " + str(resultado))
except ZeroDivisionError:
  print("O segundo número não pode ser zero")
except ValueError:
  print("Você deve inserir dois números")
finally:
  print("Programa concluído")
```

Primeiro, sem entrar em uma exceção:

```
> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: 12
> Insira o segundo número: 3
> O resultado é 4.0
> Programa concluído
```

E agora, entrando em uma exceção:

```
> Vamos dividir dois números inseridos por você
>
> Insira o primeiro número: 12
> Insira o segundo número: 0
> O segundo número não pode ser zero
> Programa concluído
```

Não importa, o código dentro do bloco "*finally*" sempre será executado. E podemos usar os dois juntos, tendo um bloco "else" e um bloco "finally" após a definição das suas condições.

Chamando Exceções

Você também pode chamar exceções em seu programa, através da declaração "*raise*". Vamos chamar uma exceção em um bloco "*try*" se o nome inserido for "*Felipe*":

```
nome = input("Qual o seu nome? ")
try:
  if nome == "Felipe":
    raise NameError("Não gostei do seu nome")
  print("Olá, %s" % nome)
except NameError:
  print("O programa não gostou do seu nome")
```

Se inserimos um nome qualquer, o bloco "try" chega até seu final e o bloco "except" não é executado. Desta forma, vemos apenas a mensagem "Olá, nome":

```
> Qual o seu nome? José
> Olá, José
```

Porém, se inserirmos "Felipe" como nome, uma exceção será chamada, conforme está definida dentro do bloco "if" pela declaração "raise", e o bloco "except" será chamado. Não veremos, então, o "print" que está definido na última linha do bloco "try", e sim a mensagem definida no bloco "except NameError":

```
> Qual o seu nome? Felipe
> O programa não gostou do seu nome
```

Conclusão

Concluímos assim o capítulo sobre exceções e tratamento de erros. Assim, seu programa pode passar informações ao usuário de forma mais elegante, prevendo os possíveis erros cometidos pelo usuário e passando mensagens informativas para o usuário caso ocorram.