Exceções

Mario Benevides

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Programação com Interfaces Gráfica

Agenda

Aula Passada: Classes

Nesta Aula: Exceções

- Objeto de Exceção
- Avisos
- Comando $raise$
- Classes de Exceção
- Mais de um $except$
- Comando $else$
- Comando $finally$
- Iteradores e Geradores

Exceções

- Quando um programa encontra dificuldades não previstas, diz-se que uma condição excepcional ou uma exceção ocorreu
  - Um erro é uma exceção mas nem toda exceção é um erro
- Para poder representar tais eventos, Python define os chamados objetos de exceção
- Se a condição excepcional não é prevista (e tratada), o programa termina com uma mensagem:

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Objetos de Exceção

- Cada exceção individual corresponde a um objeto de exceção, que por sua vez é uma instância de alguma classe de exceção
- Diz-se que o programa gerou ou levantou (raised) uma condição de exceção na forma de um objeto
- Um programa bem elaborado precisa capturar tais objetos e tratá-los para que a execução não seja abortada

Avisos

- Existem condições excepcionais menos sérias que não provocam o levantamento de um objeto de exceção, mas apenas são exibidas sob a forma de um aviso
- Por exemplo:

```
>>> import regex

Warning (from warnings module):
  File "__main__", line 1
DeprecationWarning: the regex module is deprecated; please
```

- Neste caso, o interpretador nos sinaliza que o módulo regex é antigo e que foi substituído por outro mais atualizado chamado re
- O programa não falha, mas o programador fica ciente que provavelmente deve re-escrever seu programa usando o módulo re para evitar obsolescência

Comando *raise*

- Para sinalizar a ocorrência de uma condição excepcional, pode-se usar o comando raise que tem uma das formas:
  - *raise* classe
  - *raise* classe (mensagem)
- Onde classe é uma das classes de exceção definidas pelo Python
  - Para saber todos os tipos de exceção consulte o manual
  - Se quiser uma classe genérica use a classe Exception

Exemplo: *raise*

```
  >>> raise Exception
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception

>>> raise Exception ("E agora?")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception: E agora?
```

Classes de Exceção

Classe	**Descrição**
Exception	Classe base para todas as exceções
AttrbuteError	Falha no acesso ou atribuição a atributo de classe
IOError	Falha no acesso a arquivo inexistente ou outros de E/S
IndexError	Índice inexistente de sequência
KeyError	Chave inexistente de dicionário
NameError	Variável inexistente
SyntaxeError	Erro de sintaxe
TypeError	Operador aplicado a objeto de tipo errado
ValueError	Operador aplicado objeto tipo certo mas valor errado
ZeroDivisionError	Divisão por zero

Criando uma Classe de Exceção

- Basta criar uma classe da forma habitual derivando-a da classe Exception
- Não é preciso redefinir qualquer método
- Exemplo:

```
>>> class MinhaExcecao(Exception): pass
    ...
>>> raise MinhaExcecao("E agora?")

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.MinhaExcecao: E agora?
```

Capturando Exceções

- Para capturar uma exceção possivelmente levantada por um trecho de código, pode-se usar a construção try/except:

```
try:
     Código
except Exceções:
     Código de tratamento da exceção
```

- Sendo que Exceções pode ser:
  - Classe
  - Classe,var
  - (Classe$_1$, ... , Classe$_n$)
  - (Classe$_1$, ... , Classe$_n$) as var
- Onde:
  - Classe$_1$, ... , Classe$_n$ são nomes de classes de exceção
  - var é uma variável à qual é atribuída um objeto de exceção

Exemplo 1:

```
 >>> try:
     a = eval(input("Entre com um numero:"))
     b = eval(input("Entre com outro numero:"))
     print (a, "/ ", b , "=" , a/b)
except ZeroDivisionError:
     print ("Erro, segundo numero não pode ser zero!")

Entre com um numero: 1
Entre com outro numero: 0
1 / 0 = Erro, segundo numero não pode ser zero!
```

Exemplo 2:

```
try:
     a = eval(input("Entre com um numero:"))
     b = eval(input("Entre com outro numero:"))
     print (a, "/ ", b , "=" , a/b)
except (ZeroDivisionError,TypeError):
     print ("Erro, tipo errado ou divisao por zero!")

Entre com um numero:1
Entre com outro numero:"a"
Erro, tipo errado ou divisao por zero!
```

Erro por divisão por zero ou por tipo.

Exemplo 3:

```
try:
     a = eval(input("Entre com um numero:"))
     b = eval(input("Entre com outro numero:"))
     print (a, "/ ", b , "=" , a/b)
except (ZeroDivisionError,TypeError) as e:
     print ("Erro:",e)

Entre com um numero:1
Entre com outro numero:"a"
Erro: unsupported operand type(s) for /: 'int' and 'str'
```

Coloca a mensagem de erro numa variável e.

Mais de um *except*

- É possível tratar diferentemente as diversas exceções usando duas ou mais cláusulas *except*
- Se quisermos nos prevenir contra qualquer tipo de erro, podemos usar uma cláusula *except* sem nome de classe
  - Outra opção é usar a classe Exception, que é base para todas as exceções e portanto casa com qualquer exceção
- Se não quisermos tratar um erro em uma cláusula except, podemos passá-la adiante usando o comando raise
  - Nesse caso, podemos usar um raise sem argumentos ou passar explicitamente um objeto de exceção

Exemplo 4:

```
try:
     a = eval(input("Entre com um numero:"))
     b = eval(input("Entre com outro numero:"))
     print (a, "/ ", b , "=" , a/b)
except (ZeroDivisionError):
     print ("Erro: Divisao pro zero:")
except (TypeError):
     print ("Erro, tipo errado:")
except :
     print ("Erro!!!")

Entre com um numero:1
Entre com outro numero:fg12
Erro!!!
```

Captura um erro qualquer.

Exemplo 5:

```
try:
     a = eval(input("Entre com um numero:"))
     b = eval(input("Entre com outro numero:"))
     print (a, "/ ", b , "=" , a/b)
except (TypeError):
     print ("Erro, tipo errado:")
except (Exception) as e:
     print ("Erro:" , e)
     raise

Entre com um numero:1
Entre com outro numero:f
Erro: name 'f' is not defined

Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
  File "<string>", line 1, in <module>
NameError: name 'f' is not defined
```

Captura um erro qualquer e imprimi a mensagem de erro retornada em e.

Comando *else*

É possível completar um comando *try* com uma cláusula *else* que introduz um trecho de código que só é executado quando nenhuma exceção ocorre:

```
try:
      Código
except (Exceções):
      Código de tratamento da exceção
else:
      Código executado se não ocorrem exceções
```

Exemplo: *else*

```
while True:
  try:
       a = eval(input("Entre com um numero:"))
       b = eval(input("Entre com outro numero:"))
       print (a, "/ ", b , "=" , a/b)
  except (Exception) as e:
     print ("Erro:" , e)
     print ("Tente de novo!")
  else:
     break

Entre com um numero:1
Entre com outro numero:f
Erro: name 'f' is not defined
Tente de novo!
Entre com um numero:1
Entre com outro numero:2
1 /  2 = 0.5
```

Exemplo: *else*

```
while True:
  try:
       a = eval(input("Entre com um numero:"))
       b = eval(input("Entre com outro numero:"))
       print (a, "/ ", b , "=" , a/b)
  except (Exception) as e:
     print ("Erro:" , e)
     print ("Tente de novo!")
  else:
     break

Entre com um numero:1
Entre com outro numero:f
Erro: name 'f' is not defined
Tente de novo!
Entre com um numero:1
Entre com outro numero:2
1 /  2 = 0.5
```

Repete enquanto não entra com 2 valores. No caso de erro, captura a mensagem em e.

Comando *finally*

- A cláusula *finally* pode ser usada para se assegurar que mesmo que ocorra algum erro, uma determinada seqüência de comandos vai ser executada
  - Pode ser usada para restabelecer alguma variável para um valor default, por exemplo
- A comando *finally* e comando *except* são mutuamente exclusivas
  - Neste caso exceções não são tratadas
  - É possível combinar ambas usando comandos *try* aninhados.

Exemplo: *finally*

```
try:
  try:
        x = eval(input("Entre com um numero: "))
        print (x)
  finally:
     print ("Fazendo x igual ao valor default None")
     x = None
except:
     print ("Ocorreu um Erro!" )

Entre com um numero: f
Fazendo x igual ao valor default None
Ocorreu um Erro!
```

Repare que o *finally* e o *except* estão em níveis de identação diferentes..

Iteradores

- São maneiras genéricas de implementar iterações com classes
  - Usados no comando for
  - É geralmente mais econômico do que usar uma lista pois não é preciso armazenar todos os valores, mas apenas computar um por vez
- Um iterador é uma classe que implementa o método mágico __iter__
  - É um método que, por sua vez, retorna um objeto que implementa um método chamado $__next__$
  - O método next deve retornar o "próximo" valor a ser iterado
  - Se não há próximo valor, $next$ deve "levantar" a exceção $StopIteration$

Exemplo: Iteradores

```
class MeuIterador:
        a = 0
        def __iter__(self): return self
        def __next__(self):
            if self.a > 10: raise StopIteration
            self.a += 1
            return self.a
itt = MeuIterador()
for i in itt:
    print (i , end=' ')

1 2 3 4 5 6 7 8 9 10 11
```

O *iterador* itt vai ser incrementado a cada passo. Se usamos *range* a lista fica na memória. Mais Eficiente!

Geradores

- Geradores são funções especiais que retornam iteradores
- Usa o comando $yield\ valor$
- usada para obter o $iterador$ para um comando for
- O for automaticamente iterará sobre os valores que $yield$ "retorna"
- Observe que o $iterador$ produzido pela função geradora é tal que o código que gera os valores e o código dentro do for se sucedem alternadamente

Exemplo: Geradores

```
def gerador():
  for i in range(10):
    print ("i = ", i)
    yield i
for j in gerador():
   print ("j = ",j)
```

```
i =  0
j =  0
i =  1
j =  1
....
i =  9
j =  9
```

Repare na alternância entre os comandos for.

Projeto 1: Objeto Fração

Escreva uma classe para manipular objetos do tipo fração.
Frações são números racionais da forma n/d, onde n e d são inteiros.

- A sua classe deve fazer a sobrecarga dos operadores, $+, +=, *, -, ==, /$, e $print$.
- O operador de igualdade deve simplificar as frações antes de compará-las (use o MDC).
- Levante uma exceção se algum operador for aplicado a um tipo inválido, ou se for construída uma fração com denominador nulo.

Solução do Projeto 1: Objeto Fração

Resolva o Exercício

E só então retorne para assistir a explicação da solução.

É importante que você faça o projeto e compare com a nossa solução.

E só então retorne para assistir a explicação solução.