Exceções em Python (Parte 1)

por Miguel Jonathan – DCC-IM-UFRJ

Resumo dos conceitos e regras gerais do uso de exceções em Python:

Uma exceção é um objeto da classe `Exception`, ou de uma de suas subclasses. Ele permite armazenar informações sobre situações excepcionais que venham a ocorrer durante a execução de comandos dentro de algum método.

O uso de exceções permite separar a detecção da ocorrência de uma situação excepcional dentro de um módulo, do seu tratamento (ou seja, o que fazer com a situação de erro)

Tradicionalmente, quando queremos detectar e tratar a ocorrência de um ou mais erros em alguma função `h()`, fazemos algo assim:

```
def h():
     if <ocorreu condição de erro 1>:
          <comandos que determinam o que fazer se o erro 1 ocorreu

     elif <ocorreu condição de erro 2>:
          <comandos que determinam o que fazer se o erro 2 ocorreu>

     elif <testando e tratando outros possíveis erros>

     else:
          <comandos para processar a função em condições normais>
```

Onde isso prejudica o bom design de programas?

As funções ou métodos de classes estão dentro de módulos. E podem ser usadas em aplicações diferentes.

É provável que cada aplicação precise reagir a uma mesma situação excepcional de maneira diferente.

Mas a forma acima obriga que a reação a um tipo de erro será sempre a mesma, determinada pelo bloco do if que trata esse erro.

A alternativa seria alterar a parte que trata o erro cada vez que fôssemos usar a função em uma aplicação diferente.

Mas isso obrigaria a testar toda a função de novo, a cada uso, com aumento de custos e riscos.

Além de tudo isso, a forma acima torna o código da função muito maior e complicado, dificultando a sua compreensão, complicando a documentação, e aumentando a chance de outros erros passarem despercebidos.

Como funciona o mecanismo de exceções:

O mecanismo de exceções permite escrever funções "limpas", que só se preocupam com a sua finalidade principal, sem risco de se tornarem obscuras por excesso de tratamento de erros.

Suponha uma função (digamos, para ilustrar, `f()`, mas poderia ser qualquer outra) que chama outra função `g()`:

```
def f():
     ......
     ......
     g()
     ......
     ......
```

Suponha também que, de dentro de `g()`, a função `h()` seja chamada:

```
def g():
     ......
     ......
     h()
     ......
     ......
```

Usando exceções, a função `h()` ficaria com a forma abaixo:

```
def h():
   if <ocorreu condição de erro 1>:
        raise <classe de exceção apropriada>
    elif <ocorreu a condição de erro 2> :
        raise <classe de exceção apropriada>
    etc.....
    else:
        <comandos para processar h() em condições normais sem erro>
```

Agora a função `h()` não precisa mais determinar o que fazer quando cada caso de erro ocorrer.
Ela precisa apenas detectar que o caso de erro ocorreu.
A partir daí, para cada caso de erro, ela irá construir um objeto especial de uma classe apropriada de exceção, e usará o comando *raise* para lançar (ou levantar) essa exceção para a função que a chamou.

No nosso exemplo, esse objeto poderá ser "capturado" pela função `g()` que chamou `h()`, e "tratado" lá, ou mesmo ser novamente lançado por `g()` para ser capturado e tratado por quem a chamou, no caso `f()`.

Como construir e como lançar uma exceção:

A linguagem Python já possui uma grande quantidade de classes de exceção pré-definidas. Todas descendem da classe `Exception`. Exceções são objetos comuns, instâncias da classe `Exception`, ou de alguma subclasse dela.
Para ver todas as classes de exceção, digite no Idle:

```
import exceptions
dir(exceptions)
```

Para lançar uma exceção de dentro de uma função ou método, usa-se o comando `raise <nome da classe>`.

Capturando e tratando exceções: os blocos `try e except:`

Quando programamos um módulo em Java, e dentro desse módulo existem comandos ou chamadas de funções ou métodos onde podem ocorrer exceções, os comandos devem ser colocados dentro de um bloco `try`, que tem a forma:

```
try:
  <comandos>
```

No caso de ocorrer uma exceção dentro de alguma função ou comando do bloco `try`, ela será lançada, e os demais comandos do bloco serão suspensos. O controle da execução passará para o primeiro bloco `except` que tenha uma classe de exceção de tipo compatível com a exceção lançada.
Podem existir zero, um ou mais blocos `except` após um bloco `try`.

Exemplos (bem simples):

1. No exemplo abaixo, o programa ficará rodando no laço while até que o usuário entre com 2 números, com o segundo diferente de zero:

```
fim = False
while not fim:
    try:
      a = input ("Digite um numero: ")
      b = input ("Digite outro numero: ")
      print a/b
      fim = True
    except ZeroDivisionError:
        print "Você tentou dividir por zero. Tente novamente"
    except TypeError:
        print 'Você usou um valor não numérico. Tente novamente'
```

   Note o seguinte:
   a) Se b for zero, o comando a/b lançará (levantará) automáticamente uma exceção do tipo ZeroDivisionError. Se isso ocorrer, o bloco try será imediatamente interrompido neste ponto, e o controle passará para o primeiro comando do bloco except para essa classe de exceção. Logo, o comando fim = True não chegará a ser executado e o laço não terminará, voltando o programa a solicitar os números ao usuário.
   b) Se a ou b não for do tipo de um número, o comando a/b levantará uma exceção do tipo TypeError.

   Exemplo de execução do programa acima:

```
Digite um numero: 34
Digite outro numero: 0
Você tentou dividir por zero. Tente novamente
```

```
Digite um numero: "x"
Digite outro numero: 450
Você usou um valor não numérico. Tente novamente
Digite um numero: 45.8
Digite outro numero: 2.5
18.32
```

2. Nesse segundo exemplo, uma função `somaNum(a,b)` deve retornar o valor da soma de a e b. Porém, se a for maior que 1000, a função deve levantar uma exceção do tipo `OverflowError`. A função está em um módulo chamado `lancaEx.py`:

```
#módulo lancaEx.py
def somaNum(a,b):
    if a >1000:
        raise OverflowError
    else:
        return a+b
```

Em um outro módulo, um programa utiliza essa função, dentro de um bloco `try`, e usa um bloco `except` para tratar o erro, caso ocorra:

```
from lancaEx import somaNum

a= input("Digite um número: ")
b = input ("Digite outro número: ")
try:
    c = somaNum(a,b)
    print "A soma dos dois números é",c

except OverflowError:
    print "Você digitou um valor elevado demais para o primeiro número"
```

Exemplos de uso do programa:

```
>>>
Digite um número: 78
Digite outro número: 22
A soma dos dois números é 100
>>>
Digite um número: 2500
Digite outro numero: 456
Você digitou um valor elevado demais para o primeiro número
```

3. Criando novas classes de exceção
É fácil criar classes de exceção especializadas.
Por exemplo:

```
class ExcecaoNumeroMuitoGrande(Exception):
     pass

def h(x):
    if x > 1000000:
        raise ExcecaoNumeroMuitoGrande()
    else:
        return x**2


def f():
    try:
        a = input("Digite um numero: ")
        print h(a)
    except ExcecaoNumeroMuitoGrande:
        print "Você digitou um numero maior que 1000000"
```

A cosntrução:

```
class ExcecaoNumeroMuitoGrande(Exception):
      pass
```

cria uma nova classe "derivada" da classe Exception, de nome ExcecaoNumeroMuitoGrande.

Dizemos que essa nova classe é "sub-classe" da classe Exception, e que Exception é "super-classe" dela.
Ainda não estudamos sub-classes, mas basta saber que a sub-classe possui todas as propriedades da sua super-classe (atributos e métodos), além de outras que podemos acrescentar.
Neste exemplo, a nova classe não acrescenta nenhum atributo novo, por isso o comando pass.

A função h(x) é definida para "lançar" (ou "levantar") essa exceção caso o valor de x seja maior que 1.000.000.
A função f() usa a função h(x) dentro de um bloco try, e "captura" essa exceção em um bloco except, caso ela seja lançada.