Itaú Unibanco

Itaú

Programa de formação

ITAú analytics

Módulo I – Fundamentos Computacionais
Sessão 4 - Aula 1 – Tratamento de Exceção

Prof. Dr. Luiz Alberto Vieira Dias
Prof. Dr. Lineu Mialaret

Introdução

- Exceções (*exceptions*) são eventos inesperados que ocorrem durante a execução de um programa
  - Elas podem resultar de erros lógicos ou de situações não antecipadas
- Na linguagem Python, exceções são objetos que são disparados ou levantados (*raised*) pelo interpretador que encontra uma situação não esperada
  - O interpretador pode levantar uma exceção se ele encontra uma condição não esperada, tal como falta de memória ou uma divisão por 0
- Uma exceção levantada pode ser apanhada por um contexto de código que tratará a exceção de modo apropriado
  - Caso contrário o interpretador aborta a execução e reporta a exceção

Classes de Exceções

- A linguagem Python disponibiliza de uma hierarquia de classes de exceções que simbolizam várias categorias de erros

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

Classes de Exceções (cont.)

- Exemplos:

```
>>> abs ('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: bad operand type for abs(): 'str'
```

```
>>> int ('3.14')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
'3.14'
```

```
>>> abs(3,5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: abs() takes exactly one argument
(2 given)
```

```
>>> s='12345'
>>> s[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

Instrução try

- No Python, as exceções podem ser manipuladas usando a instrução try
  - Uma instrução crítica que pode gerar uma exceção é colocada dentro da cláusula try
  - O código que manipula a exceção é inserido na cláusula except

```
>>> (x,y) = (5,0)
>>> try:
...     z = x/y
... except ZeroDivisionError:
...     print ("divide by zero")
...
divide by zero
```

Instrução try (cont.)

```
# import module sys to get the type of
exception
import sys
randomList = ['a', 0, 2]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
print("The reciprocal of",entry,"is",r)
```

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.
The entry is 0
Oops! <class 'ZeroDivisionError'> occured.
Next entry.
The entry is 2
The reciprocal of 2 is 0.5
```

Instrução try (cont.)

- Uma cláusula try pode ter qualquer número de clausulas except (com diferentes valores) para manusear as diferentes exceções
  - Mas somente uma cláusula except será executada quando uma exceção ocorrer

```
try:
    f = open('integers.txt')
    s = f.readline()
except (IOError, ValueError):
    print("An I/O error or a ValueError occurred")
except:
    print("An unexpected error occurred")
```

Raising Exceptions

- As exceções são disparadas quando um erro ocorre em tempo de execução, mas pode-se forçar o disparo da exceção usando-se a cláusula raise
  - Opcionalmente, pode-se passar valor para a exceção para clarificar o porquê a exceção foi disparada

```
>>> raise MemoryError("This is an memory error")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
MemoryError: This is an memory error
```

```
>>> try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive
number!")
... except ValueError as ve:
...     print(ve)
...
```

```
Enter a positive integer: -2
That is not a positive number!
```

Try … finally

- A instrução try pode ter uma cláusula finally opcional
  - Ela é executada independente do ocorreu antes e geralmente é utilizada para liberar recursos

```
>>> try:
...     f = open('d:/test.txt')
...     s = f.readline()
... except (IOError, ValueError):
...     print("An I/O error or a ValueError
occurred")
... except:
...     print("An unexpected error occurred")
... finally:
...     f.close()
...     print ("file closed")
...
file closed
```

Exceções Customizadas

- A linguagem Python disponibiliza muitas exceções embutidas (*built-in exceptions*) que forçam um programa a produzir um erro quando ocorre uma exceção em tempo de execução
  - Entretanto pode ser necessário criar-se exceções adaptadas para propósitos específicos (*custom exceptions*)
- Pode-se definir tais exceções criando-se uma nova classe de exceções, que é derivada (subclasse) direta ou indiretamente da classe Exception
  - A maioria das exceções do Python são derivadas desta classe

Exceções Customizadas (cont.)

- Exemplo: Criação de exceções customizadas

```
# define Python user-defined exceptions
class Error(Exception):
   """Base class for other exceptions"""
   pass

class ValueTooSmallError(Error):
   """Raised when the input value is too
small"""
   pass

class ValueTooLargeError(Error):
   """Raised when the input value is too
large"""
   pass
```

Exceções Customizadas (cont.)

- Exemplo:

```
# define Python user-defined exceptions
class Error(Exception):
   """Base class for other exceptions"""
   pass
class ValueTooSmallError(Error):
   """Raised when the input value is too
small"""
   pass
class ValueTooLargeError(Error):
   """Raised when the input value is too
large"""
   pass
```

```
Enter a number: 4
This value is too small, try again!
Enter a number: 2
This value is too small, try again!
Enter a number: 78
This value is too large, try again!
Enter a number: 10
Congratulations! You guessed it
correctly.
```

```
# our main program
# user guesses a number until he/she gets it right
# you need to guess this number
number = 10
while True:
   try:
      i_num = int(input("Enter a number: "))
      if i_num < number:
         raise ValueTooSmallError
      elif i_num > number:
         raise ValueTooLargeError
      break
   except ValueTooSmallError:
      print("This value is too small, try again!")
   except ValueTooLargeError:
      print("This value is too large, try again!")
print("Congratulations! You guessed it correctly.")
```