MC-102 — Aula 22
Exceções

Prof. Luiz F. Bittencourt

Turmas QR

Instituto de Computação – Unicamp

2019

Conteúdo adaptado de slides fornecidos pelo Prof. Eduardo Xavier.

Roteiro

1. Exceções
   - Lançando Exceções
2. Exercícios

Exceções

- Exceções são objetos utilizados em Python para avisar que algum problema ocorreu durante a execução de algum comando.
- Por exemplo, quando acessamos uma posição inválida de uma lista é gerada uma exceção **IndexError**.

```
>>> l = [1, 2, 3]
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Exceções

- Até agora tratamos exceções como eventos fatais em que o programa é encerrado.
- Na verdade, nossos programas deveriam tratar as exceções.
- Em um programa bem escrito, exceções não tratadas devem ser realmente **exceções**.
- Aprenderemos como tratar exceções.

Exceções

- Considere o trecho de código:

```
razaoSucessoFalha = num_sucessos/num_falhas
print('Razão é:', razaoSucessoFalha)

if a > b:
    .
    .
    .
```

- O código executa corretamente na maior parte do tempo.
- Quando pode ocorrer um erro?

Exceções

- Quando **num_falhas** for igual a 0, o código relativo a divisão gerará a exceção **ZeroDivisionError**.

```
razaoSucessoFalha = num_sucessos/num_falhas
print('Razão é:', razaoSucessoFalha)

if a > b:
    .
    .
    .
```

Exceções

- Se sabemos que um trecho de código pode gerar uma exceção devemos usar a construção **try-except**.
- Veja como ficaria o exemplo:

```
try:
  razaoSucessoFalha = num_sucessos/num_falhas
  print('Razão é:', razaoSucessoFalha)
except ZeroDivisionError:
  print('Razão indefinida, zero falhas')

if a > b:
    .
    .
    .
```

Exceções

```
try:
  razaoSucessoFalha = num_sucessos/num_falhas
  print('Razão é:', razaoSucessoFalha)
except ZeroDivisionError:
  print('Razão indefinida, zero falhas')

if a > b:
    .
    .
    .
```

- Se o código dentro do bloco do comando **try** gerar uma exceção, a execução do programa é alterada para o bloco **except** correspondente à exceção.
- Caso não sejam geradas exceções, após finalizado o bloco do **try** o fluxo de execução segue.
- Caso seja gerada uma exceção, após a execução do bloco do **except** o fluxo de execução segue.

Exceções

- No exemplo abaixo, caso o usuário digite algo que não seja inteiro, o programa encerrará a execução.

```
val = int(input('Digite um número'))
print('Número ao quadrado:', val**2)
```

- Como podemos fazer uma função que lê inteiros e não provoca a interrupção do programa?

Exceções

- Podemos criar a função abaixo que recebe como parâmetro a mensagem a ser impressa e devolve um número inteiro digitado.

```
def readInt(mesg):
    while True:
        val = input(mesg)
        try:
            val = int(val)
            break
        except ValueError:
            print('Formato de entrada inválido')
    return val
```

- Se o formato for inválido, na transformação para **int** teremos uma exceção, o fluxo de execução passa para o **except**, com impressão do aviso, e volta-se pro começo do laço.
- Se o formato for válido, o comando **break** será executado e o laço finalizado, e por fim o valor é devolvido.

Exceções

- Abaixo temos um exemplo com uso da função:

```
def readInt(mesg):
    while True:
        val = input(mesg)
        try:
            val = int(val)
            break
        except ValueError:
            print('Formato de entrada inválido')
    return val


def main():
    x = readInt('Número inteiro:')
    y = readInt('Número inteiro:')
    print('Soma:', x+y)

main()
```

Exceções

- Tudo em Python é um objeto, inclusive os tipos de dados como **int, float, str** etc.
- Podemos alterar a função anterior para ler dados de um tipo especificado pelo segundo parâmetro.

```
def readVal(mesg, vType):
    while True:
        val = input(mesg)
        try:
            val = vType(val)
            break
        except ValueError:
            print('Formato de entrada inválido')
    return val
```

Exceções

- Abaixo temos um exemplo com a nova função:

```
def readVal(mesg, vType):
    while True:
        val = input(mesg)
        try:
            val = vType(val)
            break
        except ValueError:
            print('Formato de entrada inválido')
    return val


def main():
    x = readVal('Número inteiro:', int)
    y = readVal('Número inteiro:', int)
    print('Soma:', x+y)

    x = readVal('Float 1:', float)
    y = readVal('Float 2:', float)
    print('Soma:', x+y)
main()
```

Lançando Exceções

- Em muitas linguagens de programação, é comum definir alguns valores em caso de erro como retorno da função.
- Exemplo: função que calcula produto interno pode devolver **None** para indicar que os vetores possuem tamanhos distintos.
- Em Python usamos exceções como retorno de função para indicar que a função não pôde devolver um valor consistente com sua definição (**int()**, por exemplo só faz sentido se o argumento estiver no formato de um número).

Lançando Exceções

- Para *lançar* uma exceção usamos o comando **raise**:

```
raise exceptionName(arguments)
```

- **exceptionName** é o nome de alguma exceção existente, seja já definida (como **ValueError**) ou definida por você como sub-classe de **Exception**.
- Os argumentos dependem da exceção mas a maioria aceita uma string com descrição do problema.

Lançando Exceções

- Considere o exemplo de função:

```
def produtoInterno(v1, v2):
    if len(v1) != len(v2):
        raise ValueError('Dimensões diferentes para o calculo')

    sum = 0
    for i in range(len(v1)):
        sum = sum + v1[i]*v2[i]

    return sum
```

- A função verifica se os vetores possuem dimensões compatíveis para o cálculo; caso não tenham, lança-se uma exceção.

Lançando Exceções

- Considere o exemplo:

```
import random

def produtoInterno(v1, v2):
    if len(v1) != len(v2):
        raise ValueError('Dimensões diferentes para o calculo')

    sum = 0
    for i in range(len(v1)):
        sum = sum + v1[i]*v2[i]

    return sum


def main():
    l1 = [random.randint(0,10) for i in range(5)]
    l2 = [random.randint(0,10) for i in range(4)]

    print('v1:', l1)
    print('v2:', l2)

    try:
        print(produtoInterno(l1,l2))
    except ValueError as msg:
        print(msg)

main()
```

- O que será impresso?

Lançando Exceções

- Uma função pode lançar várias exceções distintas, dependendo do tipo de problema.
- A nova versão da função também verifica se os argumentos são listas:

```
def produtoInterno(v1, v2):
    if type(v1) != list or type(v2) != list:
        raise TypeError('Operação definida apenas para listas')

    if len(v1) != len(v2):
        raise ValueError('Dimensões diferentes para o calculo')

    sum = 0
    for i in range(len(v1)):
        sum = sum + v1[i]*v2[i]

    return sum
```

Lançando Exceções

- Para cada possível exceção, o invocador da função terá uma cláusula **except**:

```
try:
    print(produtoInterno(l1,l2))
except ValueError as msg:
    print(msg)
except TypeError as msg:
    print(msg)
```

Lançando Exceções

```
import random

def produtoInterno(v1, v2):
    if type(v1) != list or type(v2) != list:
        raise TypeError('Operação definida apenas para listas')

    if len(v1) != len(v2):
        raise ValueError('Dimensões diferentes para o calculo')

    sum = 0
    for i in range(len(v1)):
        sum = sum + v1[i]*v2[i]

    return sum


def main():
    l1 = [random.randint(0,10) for i in range(5)]
    l2 = [random.randint(0,10) for i in range(5)]

    l1 = 5

    try:
        print(produtoInterno(l1,l2))
    except ValueError as msg:
        print(msg)
    except TypeError as msg:
        print(msg)

main()
```

- O que será impresso?

Exercício 1

Anteriormente implementamos funções para operações sobre matrizes:

- **def soma(m1, m2)**: devolve **m3** que é a soma das matrizes parâmetros.
- **def multiplica(m1, m2)**: devolve **m3** que é a multiplicação das matrizes parâmetros.

Refaça as funções para que devolvam exceções apropriadas caso as matrizes passadas como parâmetro não possuam dimensões compatíveis com a operação pretendida.