Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa
DEPARTAMENTO·DE·INFORMÁTICA

Folha 10 - Exceções, asserções

Python 3.*

Exercícios

1) Considere o seguinte excerto de uma interação com o interpretador de Python:

```
x = int(input("Número: "))
Número: olá

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x = int(input("Número: "))
ValueError: invalid literal for int() with base 10: 'olá'
```

   a) O que aconteceu e porquê?

   b) Identifique o erro. Qual a mensagem associada ao erro?

2) Considere o seguinte excerto de código contido no ficheiro `ler_inteiros.py`

```
def ler_numero(mensagem):
    return int(input(mensagem))

print(ler_numero("Número: "))
```

   e a execução

```
Número: olá

Traceback (most recent call last):
  File "/Users/fmartins/Desktop/ler_inteiros.py", line 4, in <module>
    print ler_numero("Número: ")
  File "/Users/fmartins/Desktop/ler_inteiros.py", line 2, in ler_numero
    return int(input(mensagem))
ValueError: invalid literal for int() with base 10: 'olá'
```

   a) Analise o *Traceback* do programa. Em que função e em que linha ocorreu o erro?

   b) De que linha a função foi chamada.

3) Considere o seguinte excerto de código contido no ficheiro `ler_dados.py`

```
def ler_numero(mensagem):
    return int(input(mensagem))
```

e no ficheiro `ler_positivo.py`

```
from ler_dados import ler_numero

def ler_inteiro_positivo(mensagem):
    x = ler_numero(mensagem)
    while x <= 0:
        x = ler_numero(mensagem)
    return x

print(ler_inteiro_positivo("Número: "))
```

e a execução

```
Número: olá

Traceback (most recent call last):
  File "/Users/fmartins/Desktop/ler_positivo.py", line 9, in <module>
    print ler_inteiro_positivo("Número: ")
  File "/Users/fmartins/Desktop/ler_positivo.py", line 4, in
ler_inteiro_positivo
    x = ler_numero(mensagem)
  File "/Users/fmartins/Desktop/ler_dados.py", line 2, in ler_numero
    return int(input(mensagem))
ValueError: invalid literal for int() with base 10: 'olá'
```

a) Identifique em que ficheiro e linha ocorreu o erro.

b) Analise o *Traceback* do programa.

4) Considere a seguinte função:

```
def f(iteravel):
    v1 = 0.0
    for v2 in iteravel:
        v1 += v2
    return v1 / len(iteravel)
```

a) O que calcula a função `f`?

b) O que devolve a função caso o iterável esteja vazio?

c) Altere a função de forma a que esta lance a exceção `ValueError` quando o `iteravel` estiver vazio e forneça uma mensagem explicativa adequada.

5) Considere o seguinte excerto de uma interação com o interpretador de Python:

```
try:
    x = int(input("Número: "))
    print(2 * x)
except ValueError:
    print("O valor introduzido não corresponde a um número")
print("a seguir ao bloco")
```

a) Identifique o bloco de instruções que constitui a cláusula do `try`.

b) O que é escrito no ecrã caso o utilizador introduza a palavra `olá`?

c) Explique o fluxo de execução do programa neste caso.

d) Qual a saída do programa caso o utilizado introduza o número `14`.

e) Explique o fluxo de execução do programa neste segundo caso.

6) Considere o seguinte excerto de código:

```
try:
    fic = open('ficheiro.txt')
    linha = fic.readline()
    numero = int(linha.strip())
    fic.close()
except IOError as e:
    print("Erro de entrada/saída ({0}): {1}."\
        .format(e.errno, e.strerror))
except ValueError:
    print("Número inteiro invalido.")
else:
    print("Programa executado sem problemas.")
```

a) Quais as exceções que são monitorizados pela cláusula do `try`.

b) Qual o papel da variável `e` no tratamento da primeira exceção.

c) Indique o fluxo do programa que leva a imprimir a linha associada à exceção `IOError`.

d) E qual o fluxo que leva a imprimir a mensagem de erro associada à exceção `ValueError`.

e) Em que circunstâncias é que o bloco de código associado ao `else` é executado?

7) Considere uma variante do excerto de código do exercício anterior:

```
try:
    fic = open('ficheiro.txt')
    linha = fic.readline()
    numero = int(linha.strip())
    fic.close()
except (IOError, ValueError) as e:
    print("Erro ao processar o ficheiro ({0}): {1}."\
          .format(e.errno, e.strerror))
finally:
    print("Programa executado.")
```

a) Analise o bloco `except`. Quais são as exceções que o bloco monitoriza? Qual a diferença para o bloco do exercício anterior? Quando se pode usar uma ou outra forma?

b) Em que circunstâncias é que o bloco de código associado ao `finally` é executado? Indique o que o programa imprime quando é e quando não é lançada qualquer exceção no bloco associado ao `try`.

8) Considere a seguinte função que adiciona os inteiros entre `a` e `b`:

```
def soma_ab(a, b):
    """
    Soma os números inteiro entre a e b inclusivamente.
    Requires: a e b int > 0 e a <= b
    Ensures: a + (a+1) + (a+2) + ... + b
    """
    assert a > 0
    assert b > 0
    c = a
    s = 0
    while c < b:
        s += c
        c += 1
    assert s == (a+b)*(b-a+1)/2
    return s
```

a) Qual o significado dos comandos `assert`?

b) Qual o resultado da execução da função quando invocada com os argumentos `-1`, `3`?

c) E quando executada com os argumentos `2`, `-3`?

d) E ainda com `1`, `5`?

e) Como corrigir o programa para a asserção referente à pós-condição não falhar?

9) Considere a seguinte função que procura um elemento numa lista.

```
def procura(lista, valor):
    i = 0
    while i < len(lista) and lista[i] != valor:
        i += 1
    try:
        assert i < len(lista)
        return i
    except:
        raise KeyError("O valor não foi encontrado na lista")
```

a) O que devolve a função quando invocada com os argumentos `[5,3,1,2]` e `2`?

b) E quando invocada com os argumentos `[5,3,1,2]` e `4`? Qual o erro e a mensagem escritas no ecrã?

Problemas

1) Escreva uma função que receba por parâmetro um iterável e que determine o elemento que se repete mais vezes. A função deve devolver um par contendo o elemento mais frequente e o número de vezes que ele se repete. Para tal utilize um dicionário para contar as ocorrências dos vários elementos e depois determinar o maior.

2) Elabore uma versão da função do problema anterior que recorra ao tratamento de exceções para determinar que um elemento não pertence ao dicionário. Use o interpretador para identificar a exceção que é lançada quando uma chave não faz parte do dicionário. Elabore um programa que exercite a função desenvolvida.

3) Escreva uma função leitura que recebe uma *string*, efetua uma leitura do teclado (entrada padrão) afixando a mensagem recebida e devolva o valor lido. Em caso da entrada de valores ser interrompida, por exemplo, premindo `control-c`, a função deve lidar com a exceção correspondente e devolver a *string* vazia.

4) Implemente uma função `encontra` que recebe um iterável e um elemento e devolva um tuplo com as posições em que se encontra o elemento. Caso o elemento não exista, deve ser lançada a exceção `KeyError` com uma mensagem de texto adequada.

5) Escreva uma função `ler_inteiro` que lê, garantidamente, um inteiro do utilizador. A função deve pedir a informação ao utilizador até que seja fornecido um número inteiro. Faça um programa que teste convenientemente a função desenvolvida.

6) Escreva uma função que recebe uma *string* com o nome de um ficheiro que deveria conter um número inteiro em cada linha e devolva a soma destes números. Caso o ficheiro não exista, a função deve devolver zero. As linhas que não têm números inteiros devem ser ignoradas.

7) Escreva uma função que recebe um dicionário e devolve outro dicionário com a relação inversa, ou seja, para cada par `chave:valor` do dicionário recebido por parâmetro, o dicionário resultado deve conter uma entrada `valor:chave`. Caso a relação não tenha inversa, ou seja, quando existem várias chaves associadas ao mesmo valor no dicionário recebido, a função deve lançadar uma exceção `ValueError` com uma mensagem conveniente. Elabore um programa que teste a sua função para os diversos casos.

8) Escreva uma função `media` que recebe um tuplo de números pares e devolve a sua média.
   a) Escreva a *docstring* com o contrato da função;
   b) Tente escrever comandos `assert` que traduzem os contratos escritos em texto;
   c) Defina uma função que verifique se um tuplo é constituído só por números pares;
   d) Escreva outra função que some os elementos de um tuplo;
   e) Rescreva a função `media` usando as funções definidas na alínea c) e d);
   f) Escreva agora os comandos `assert` que traduzem os contratos da função `media`;
   g) Faça um programa que teste de forma conveniente as funções desenvolvidas, em particular, inclua casos que testem o funcionamento correto e a falha das pré-condições.
   h) Introduza um erro na função `soma` de forma a fazer falhar a pós-condição da função `media`.