Programação em Python

Manipulação de ficheiros e Gestão de Exceções

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Ficheiros

Ficheiros

- Agregado de dados coerente; por exemplo:
  - uma documento de texto;
  - uma faixa de música;
  - o código-fonte de um programa Python.
- Identificado por um *caminho* no sistema de ficheiros
- Ao contrário das variáveis dum programa: os ficheiros são persistentes
- Suportes físicos: discos magnéticos, memórias flash, CD-Rs, . . .

Manipular ficheiros em Python

Três fases:

1. abrir o ficheiro
2. ler e/ou escrever no ficheiro
3. fechar o ficheiro

Manipular ficheiros em Python

Acesso por meio de objectos de tipo `file`:

```
>>> f = open("test.dat","w")
>>> type(f)
<class '_io.TextIOWrapper'>
>>> f
<_io.TextIOWrapper name='test.dat' mode='w'
 encoding='UTF-8'>
```

Abrir um ficheiro

```
f = open(caminho, modo)
```

Modos:

`'r'`	leitura (ficheiro deve já existir)
`'w'`	escrita (se o ficheiro já existir: remove o conteúdo)
`'a'`	escrita (se o ficheiro já existir: acrescenta ao final)
`'w+'`	leitura e escrita

Métodos sobre ficheiros

Mais comuns:

`f.write(str)` escrever uma cadeia de caracteres

`f.read()` lê todo o conteúdo (uma cadeia de caracteres)

`f.read(n)` lê apenas *n* caracteres

`f.readline()` lê uma linha de texto

`f.close()` terminar leitura/escrita no ficheiro

Importante: deve sempre usar `close` para garantir que a operação (escrita/leitura) termina corretamente!

Exemplo de escrita

O programa

test.py

```
f = open("test.dat", "w")
f.write("Olá mundo!\n")
f.write("Adeus mundo...\n")
f.close()
```

produz o ficheiro seguinte:

test.dat

```
Ola mundo!
Adeus mundo...
```

Exemplo de leitura

```
>>> f = open("test.dat", "r")
>>> txt = f.read()
>>> txt
'Ola mundo!\nAdeus mundo...\n'
>>> f.close()
```

Se o ficheiro não existe

```
>>> f = open("test.cat", "r")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IOError: [Errno 2] No such file or directory:
  'test.cat'
```

Ficheiros de texto ou binários

Ficheiros de texto contêm códigos de caracteres num dado sistema de codificação (por exemplo: ASCII, ISO-LATIN-1, UTF-8)

Ficheiros binários codificados de forma específica para cada uso (por exemplo: imagens JPEG, áudio MP3, binários ELF 32-bits)

- Por omissão: `open` assume que o ficheiro é de texto
- Para usar modo binário: `open(fich, "rb")` ou `open(fich, "wb")`
- Nestas aulas vamos usar apenas ficheiros de texto

Procurar ocorrências de uma palavra

Uma função para procurar uma palavra num ficheiro:

- imprime as linhas em que palavra ocorre
- versão simplificada do comando `grep` de UNIX

Exemplo de uso

```
>>> procurar("armas", "lusiadas.txt")
As armas e os barões assinalados,
Na qual vos deu por armas, e deixou
Fizeram, só por armas tão subidos,
Por armas têm adargas o terçados;
Mostra das fortes armas de que usavam,
De mim, da Lei, das armas que trazia.
Se as armas queres ver, como tens dito,
Dá-lhe armas o furor desatinado.
```

Procurar ocorrências de uma palavra

```
def procurar(palavra, ficheiro)
    ''' Imprime as linhas dum ficheiro em
    que ocorre uma palavra.'''
    f = open(ficheiro, "r")
    while True: # repetir
        linha = f.readline()  # lê uma linha
        if linha == '':       # fim do ficheiro?
            break
        if palavra in linha:  # palavra ocorre na linha?
            print(linha)
    # fim do ciclo
    f.close()
```

Tabelar uma função

Vamos agora escrever um programa que escreve um ficheiro com uma tabela de valores calculados.

Exemplo: tabelar a função $f(x) = x \sin(x)$ no intervalo $[-10, 10]$.

Programa

```
import math

def fun(x):
    return x*math.sin(x)

a = -10   # limite esquerdo
b = 10    # limite direito
n = 500   # número de sub-intervalos
file = open("tabela.txt", "w")
dx = (b-a)/n  # distancia entre pontos
x = a
for i in range(n):
    file.write("%f\t%f\n" % (x,fun(x)))
    x += dx
file.close()
```

Execução

Produz um ficheiro com 501 linhas:

tabela.txt

```
-10.000000         -5.440211
-9.960000          -5.079919
-9.920000          -4.714252
-9.880000          -4.343858
...
9.800000          -3.591495
9.840000          -3.969388
9.880000          -4.343858
9.920000          -4.714252
9.960000          -5.079919
```

Podemos importar para uma folha de cálculo e traçar um gráfico.

Gestão de Exceções

Exceções

- Durante a execução podem ocorrer vários erros:
  - divisão por zero, raíz quadrada de um número negativo
  - abrir um ficheiro que não existe
  - erro de tipos. . .
- Um erro faz lançar uma exceção
- Uma exceção causa a terminação do programa com uma mensagem de erro
- Para tornar um programa mais robusto, podemos lançar e apanhar as exceções

Apanhar exceções

```
try:
    # código que poderá lançar um erro
    ⋮
except Exceção:
    # código em caso de erro
    ⋮
```

Algumas exceções pré-definidas:

IOError	leitura/escrita de ficheiros
ValueError	argumento fora do domínio e.g. `sqrt(-1)`
IndexError	índice fora de limites
TypeError	erro de tipos

Exemplo: conteúdo de um ficheiro

```
def conteudo(fich):
    "Lê o conteúdo de um ficheiro, se existir."
    try:
        # tenta abrir o ficheiro
        f = open(fich, "r")
        # obtém o conteúdo e fecha
        cont = f.read()
        f.close()
    except IOError:
        # erro: o ficheiro não existe
        # conteudo é vazio
        cont = ''
    return cont
```

Lançar exceções

- Podemos forçar uma exceção explicitamente:

  `raise` *exceção*

- Útil em situações em que o programa não deve continuar
- Evita confundir um erro com um valor correto

Exemplo: factorial

```
def factorial(n):
  "Calcula o factorial de n."
   # lança exceção se n<0
   if n<0:
      raise ValueError("argumento negativo")
   # caso contrário: argumento é positivo
   p = 1
   while n>0:
      p = p*n
      n = n-1
   return p
```

Exemplo: factorial

```
>>> factorial(5)
120
>>> factorial(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 4, in factorial
ValueError: argumento negativo
```