Programação I
Aula 13 — Operações sobre ficheiros

Pedro Vasconcelos
DCC/FCUP

Nesta aula

1. Ficheiros
   - Métodos sobre ficheiros
   - Ficheiros de texto ou binários
   - Procurar ocorrências de uma palavra
   - Tabelar uma função
2. Exceções
   - Apanhar exceções
   - Lançar exceções

Ficheiros

- Agregado coerente de informação, e.g.:
  - uma documento de texto;
  - o código-fonte de um programa Python;
  - uma imagem captada por uma camera digital.
- Identificado por um *caminho* no sistema de ficheiros
- Ao contrário das variáveis dum programa: os ficheiros são persistentes
- Suportes físicos: discos magnéticos, SSDs, memória flash . . .

Operar com ficheiros

Três fases:

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

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

Importante: deve sempre usar `close` para garantir que o ficheiro é escrito correctamente!

O resultado de `open` é um objecto de tipo `file`:

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

Usamos métodos para operar com o ficheiro.

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 sequências de códigos de carateres (e.g. ASCII, ISO-LATIN-1, UTF-8)

Ficheiros binários formatos especializados (e.g. JPEG para imagens, MP3 para audio, ELF para executáveis)

- Por omissão: `open` assume que o ficheiros é de texto
- Nestas aulas vamos usar apenas ficheiros de texto

Codificação de texto

- O sistema de codificação de texto mais comum na *web* e em sistemas Linux é o UTF-8
  - permite texto com carateres de todas as linguas europeias, árabe, hebriaco e outras
- Em Windows: a codificação de texto pode não ser UTF-8 por omissão
- Para garantir que abrimos ficheiros usando UTF-8 podemos indicar a codificação explicitamente:

```
f = open(caminho, modo, encoding="utf-8")
```


Procurar ocorrências de uma palavra

Uma função para procurar uma palavra num ficheiro:

- imprime o número e a linha em a palavra ocorre
- versão simplificada do comando `grep` de UNIX

Exemplo de uso

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


Procurar ocorrências de uma palavra

```
def procurar(palavra, fich):
    "Ocorrências duma palavra num ficheiro."
    f = open(fich, "r", encoding="utf-8")
    n = 1        # contador de linhas
    while True: # repetir
        linha = f.readline()  # uma nova linha
        if linha == "":      # fim do ficheiro?
            break
        if palavra in linha:  # a palavra ocorre?
            print("%d:%s" % (n,linha))
        n = n + 1            # mais uma 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
dx = (b-a)/n  # distância entre pontos

f = open("tabela.txt", "w")
x = a
for i in range(n):
    f.write("%f\t%f\n" % (x,fun(x)))
    x += dx
f.close()
```

Execução

Produz um ficheiro com 501 linhas:

tabela.txt

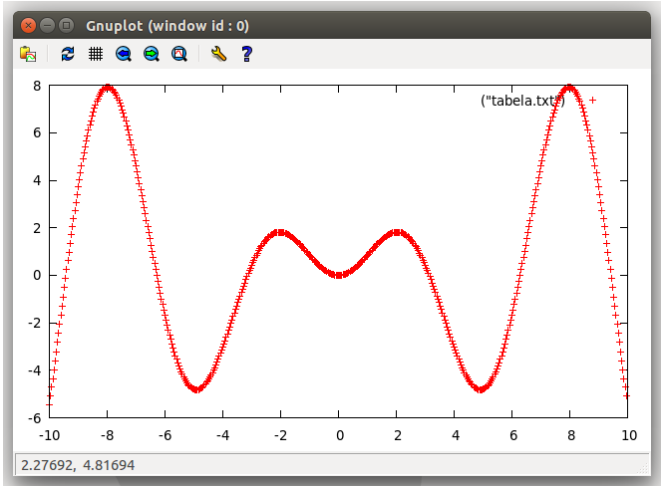
```
-10.000000        -5.440211
-9.960000         -5.079919
-9.920000         -4.714252
-9.880000         -4.343858
...
9.800000          -3.591495
9.840000          -3.969388
9.880000          -4.343858
9.920000          -4.714252
9.960000          -5.079919
```


Traçar um gráfico

Podemos usar o programa especial *Gnuplot* para traçar um gráfico.

No interpretador de comandos (*shell* de Linux):

```
$ cd diretorio do ficheiro .txt
$ gnuplot
gnuplot> plot("tabela.txt")
```



Exceções

- Durante a execução de um programa podem ocorrer difentes erros, e.g.:
  - erro de índices, erro de tipos
  - divisão por zero, raíz quadrada de um número negativo
  - abrir um ficheiro que não existe
- O erro desencadeia uma exceção
- Normalmente a excepção faz com que o programa termine com uma mensagem de erro
- Para tornar um programa mais robusto, o programador por *lançar* e *apanhar* as excepções explicitamente

Apanhar excepções

```
try:
    # código que poderá lançar um erro
    ⋮
except Excepção:
    # código de tratamento do erro
    ⋮
```

Algumas exceções pré-definidas:

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

Exemplo: conteúdo de um ficheiro

Definir uma função que lê e retorna o conteúdo de um ficheiro; caso contrário retorna a cadeia vazia.

Exemplo: conteúdo de um ficheiro (cont.)

```
def conteudo(fich):
    "Retorna 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 correcto

Exemplo: factorial

- Vamos definir uma função que calcula o factorial de um inteiro não-negativo
- Se o argumento for negativo lança uma exceção apropriada

Exemplo: factorial (cont.)

```
def factorial(n: int) -> int:
    "Calcula o factorial de um inteiro não-negativo."
    if n<0:
        raise ValueError("argumento negativo")
    # caso contrário: calcular o fatorial
    r = 1
    while n>1:
        r = r*n
        n = n-1
    return r
```

Note que, tal como `return`, o comando `raise` termina a execução da função.

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
```