Programação I

Licenciatura em Engenharia Informática

2015–2016

Vitor Beires Nogueira

Escola de Ciências e Tecnologia
Universidade de Évora

Persistência

- Os programas que vimos até agora não são "persistentes", isto é, executam durante algum tempo, recebem input, produzem output mas quando terminam a sua informação desaparece.
- Existem outro tipo de de programas denominados de persistentes:
  - executam durante "bastante tempo"(ou para sempre)
  - guardam alguma da sua informação em algum armazenamento permanente
  - se fizermos "restart", os programas retomam aonde tinham ficado
- Exemplos de programas "persistentes"
  - SO
  - web servers
- Vamos guardar o estado do programa num ficheiros ficheiro/base de dados.

Lendo e escrevendo

Para escrever para um ficheiro podemos abri-lo em modo de escrita:

```
>>> fout = open('output.txt', 'w')
>>> print(fout)
<open file 'output.txt', mode 'w' at 0xb7eb2410>
```

- Se o ficheiro não existir, é criado
- Senão, apaga o conteúdo do ficheiro e começa com o ficheiro vazio

Para colocar informação nos ficheiros, temos o método `write`:

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
```

- O "objecto"ficheiro regista o local onde está, pelo que escrevermos novamente, adiciona a nova informação a seguir.
- Quando terminarmos a escrita, devemos fechar o ficheiro `fout.close()`

O operador format

Como o argumento de write tem de ser uma string, antes de escrevermos alguma coisa temos de converter para string, como por exemplo:

```
>>> x = 52
>>> f.write(str(x))
```

Em alternativa podemos utilizar o operador "format" `%` [1].

- O primeiro operando é a "format string"(especifica como o 2o operando deve ser formatado)

[1] Este operador quando utilizado com inteiros é o módulo.

Exemplos do operador format


```
>>> camels = 42
>>> '%d' % camels
'42'
>>> camels = 42
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: illegal argument type for built-in operation
```

Nomes de ficheiros e caminhos I

Podemos inquirir qual a directoria actual:

```
>>> import os
>>> cwd = os.getcwd()
>>> print(cwd)
/home/vbn
```

Qual o caminho absoluto de um ficheiro

```
>>> os.path.abspath('memo.txt')
'/home/vbn/memo.txt'
```

Se o ficheiro/directoria existe

```
>>> os.path.exists('memo.txt')
True
```

Se é uma directoria e se é um ficheiro:

Nomes de ficheiros e caminhos II


```
>>> os.path.isdir('memo.txt')
False
>>> os.path.isfile('memo.txt')
True
```

Qual o conteúdo de uma directoria:

```
>>> os.listdir(cwd)
['Music', 'Pictures', 'Documents', 'memo.txt']
```

Exemplo

```
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            print(path)
        else:
            walk(path)
```

Excepções: exemplos

Ao lidar com ficheiros muitas coisas podem falhar:

- Podemos tentar abrir um ficheiro que não existe

```
>>> fin = open('bad_file')
IOError: [Errno 2] No such file or directory: 'bad_file'
```

- Podemos não ter permissões para abrir o ficheiro

```
>>> fout = open('/etc/passwd', 'w')
IOError: [Errno 13] Permission denied: '/etc/passwd'
```

- Podemos tentar abrir uma directoria para leitura

```
>>> fin = open('/home')
IOError: [Errno 21] Is a directory
```

Excepções: como lidar

- Podemos tentar evitar que as excepções ocorram
- Também podemos "capturar"as excepções:

```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError:
    print("I/O error")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

Pipes

- Qualquer programa que possa ser lançada a partir da shell também pode ser lançado a partir do Python utilizando um **pipe**: objecto que representa um objecto que está a "correr"
- Exemplo:

```
>>> cmd = 'ls -l'
>>> fp = os.popen(cmd)
>>> res = fp.read()
>>> stat = fp.close()
>>> print(stat)
None
```

- O resultado de `popen` pode ser tratado como `open`
- O resultado do método `close` de *pipe* indica se o processo terminou normalmente (nesse caso devolve `None`)

Exemplo: descomprir incrementalmente

```
def open_gunzip(filename):
    cmd = 'gunzip -c ' + filename
    fp = os.popen(cmd)
    return fp
```

Módulos

- Qualquer ficheiro que tenha código Python pode ser importado como um módulo.
- Considere que definimos o seguinte código no ficheiro `wc.py`:

```
def linecount(filename):
    count = 0
    for line in open(filename):
        count += 1
    return count

print(linecount('wc.py'))
```

  - Se o executarmos, ele irá ler-se a si próprio e depois escrever o número de linhas no ficheiro (7)
  - Se o importarmos, irá fazer algo semelhante:

```
>>> import wc
7
>>> print(wc)
<module 'wc' from 'wc.py'>
>>> wc.linecount('wc.py')
7
```

Variável __main__

- Um "defeito" do módulo anterior é que ao ser importado, executava a função `linecount`
- Podemos modificar a invocação no final do módulo para:

```
if __name__ == '__main__':
    print(linecount('wc.py'))
```

- A variável (buil-in) `__main__` é definida quando o programa começa.
- Se o programa estiver a ser executado como um script, a variável `__name__` fica com o valor `__main__`.