Programação I

Licenciatura em Engenharia Informática

2015–2016

Vitor Beires Nogueira

Escola de Ciências e Tecnologia
Universidade de Évora

Números Aleatórios

- Os programas de computador são *determinísticos*: com o mesmo input produzem o mesmo output.
- No entanto por vezes podemos querer programas com *comportamento imprevisível*.
- Não-determinismo verdadeiro não é fácil
- Para obter um comportamento quase não-determinístico podemos utilizar os números (pseudo) aleatórios.

Módulo random

- O módulo *random* fornece um conjunto de funções para gerar números (pseudo) aleatórios.
  - a função `random` devolve um float aleatório entre 0.0 (inclusivé) e 1.0 (exclusivé), i. e. $[0.0, 1.0[$
  - a função `randint` recebe os parâmetros `low` e `high` e devolve um inteiro aleatório entre `low` (inclusivé) e `high` (inclusivé), i. e. $[low, high]$
  - a função `choice` escolhe um elemento de uma sequência.

Módulo random: exemplos

Exemplo (random)

```
import random

for i in range(10):
    x = random.random()
    print( x)
```

Exemplo (randint e choice)

```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
>>> t = [1, 2, 3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

Histograma de palavras

Escreva um programa que leia um ficheiro de texto e construa o histograma das palavras do ficheiro. Deve ignorar/remover todos os espaços em branco assim como as pontuações.

Strings auxiliares

No módulo string:

- string `whitespace`: contém espaço, tab, newline, etc.
- string `punctuation`: `...`

Métodos das strings I

- str.replace(old, new[, count]): Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced
- str.strip([chars]) Return a copy of the string with the leading and trailing characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a prefix or suffix; rather, all combinations of its values are stripped.
- str.lower() Return a copy of the string converted to lowercase.
- str.split([sep[, maxsplit]]): Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified, then there is no limit on the number of splits (all possible splits are made).

  If sep is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, '1,,2'.split(',') returns ['1', '', '2']). The sep argument may consist of multiple characters (for example, '1 <> 2 <> 3'.split(' <>') returns ['1',' 2',' 3']). Splitting an empty string with a specified separator returns [''].

  If sep is not specified or is None, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end

Métodos das strings II

if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a None separator returns [].

For example, ' 1 2 3 '.split() returns ['1', '2', '3'], and ' 1 2 3 '.split(None, 1) returns ['1', '2 3 '].

Text File Histogram I

```
import string

def process_file(filename):
    h = dict()
    fp = open(filename)
    for line in fp:
        process_line(line, h)
    return h

def process_line(line, h):
    line = line.replace('-', ' ')

    for word in line.split():
        word = word.strip(string.punctuation +
                          string.whitespace)
        word = word.lower()

        h[word] = h.get(word, 0) + 1
```

Histogram

Como contabilizar o número total de palavras?

total_words

```
def total_words(h):
    return sum(h.values())
```

Como contabilizar o número de palavras diferentes?

different_words

```
def different_words(h):
    return len(h)
```

Palavras mais comuns

most_common

```
def most_common(h):
    t = []
    for key, value in h.items():
        t.append((value, key))

    t.sort(reverse=True)
    return t
```

Palavras mais comuns: Emma (Jane Austen)

10 mais comuns

```
t = most_common(hist)
print('The most common words are:')
for freq, word in t[0:10]:
    print(word, '\t', freq)
```

```
The most common words are:
to      5242
the     5204
and     4897
of      4293
i       3191
a       3130
it      2529
her     2483
was     2400
she     2364
```

Parâmetros opcionais

print_most_common

```
def print_most_common(hist, num=10):
    t = most_common(hist)
    print('The most common words are:')
    for freq, word in t[0:num]:
        print(word, '\t', freq)
```

Exemplo

```
>>> print_most_common(hist)
...
>>> print_most_common(hist, 20)
...
```

Subtracção de dicionários

Considere que pretendemos encontrar as palavras que estão num determinado ficheiro mas não estão num outro.

subtract

```
def subtract(d1, d2):
    res = dict()
    for key in d1:
        if key not in d2:
            res[key] = None
    return res
```

Exemplo

```
words = process_file('words.txt')
diff = subtract(hist, words)
```

Palavras aleatórias

Considere que pretendemos escolher aleatoriamente palavras de um histograma, mas que devem ser escolhidas com uma probabilidade de acordo com a sua frequência.

random_words

```
def random_word(h):
    t = []
    for word, freq in h.items():
        t.extend([word] * freq)

    return random.choice(t)
```

Análise de Markov I

- Se escolhermos aleatoriamente palavras de um livro, provavelmente não iremos obter uma frase:

  this the small regard harriet which knightley's it most things

- Numa frase correcta, provavelmente esperávamos que o artigo "the" fosse seguido de um nome ou de um adjectivo e não de verbo ou advérbio.
- A análise de Markov permite-nos saber para uma determinada sequência de palavras, a probabilidade da palavra que vem a seguir.

Análise de Markov II

- Consideremos a letra da música "Eric, the Half a Bee":

  Half a bee, philosophically,
  Must, ipso facto, half not be.
  But half the bee has got to be Vis a vis, its entity.
  D'you see?
  But can a bee be said to be
  Or not to be an entire bee
  When half the bee is not a bee
  Due to some ancient injury?

  - As palavras "half the" são sempre seguidas pela palavra "bee"
  - As palavras "the bee" podem ser seguidas por "has" ou "is"
  - O resultado de uma análise de Markov é uma atribuição de cada prefixo (como "half the" ou "the bee") aos possíveis sufixos ("bee", "has" ou "is")
  - A partir da análise de Markov podemos gerar textos interessantes.

- Que estruturas de dados utilizar para a análise de Markov? Ou seja, como representar a informação?
  - os prefixos?
    - strings?
    - lista de strings?
    - tuplos de strings?
  - a colecção de possíveis sufixos?
    - lista?
    - histograma (dicionário)?
  - a atribuição de cada prefixo a uma colecção de possíveis sufixos?
    - Simples pois ainda só conhecemos uma atribuição (dicionários)
- O tipo de operações que pretendemos efectuar condicionam a escolha mas existem outros factores:
  - tempo
  - espaço

Debugging

Estratégia dos 4 erres:

- reading: examinem o código.
- running: experimentem o código fazendo pequenas alterações.
- ruminating: pensem durante algum tempo.
- retreating: em certas alturas o melhor mesmo é voltar atrás, eliminando alterações recentes, até voltar a um programa que funcione e que entendam.