

Programação II

Formatos de dados

Hugo Pacheco

DCC/FCUP

20/21

Dados

- Nos ficheiros que temos visto até agora (extratos de livros), os dados estão em texto livre
- Mas têm sempre alguma estrutura, por exemplo, hierarquia de cantos e estrofes nos lusíadas

“Canto Primeiro

1

*As armas e os barões assinalados,
Que da ocidental praia Lusitana,
Por mares nunca de antes navegados,
Passaram ainda além da Taprobana,
Em perigos e guerras esforçados,
Mais do que prometia a força humana,
E entre gente remota edificaram
Novo Reino, que tanto sublimaram;
[...]*”

Dados

- Para ler os dados temos que explorar a sua estrutura.
- Mas se estão em texto livre...
 - pode ser difícil encontrar padrões
 - cada caso é um caso
- Idealmente, diferentes conjuntos de dados devem seguir uma estrutura comum:
 - facilitar a partilha
 - garantir preservação e suporte de longo prazo
 - melhores ferramentas

“The Web is Agreement”



Formatos de dados

- Alguns dos formatos de dados mais comuns:
 - **Comma Separated Value (CSV)**
 - Excel (XLS)
 - **JavaScript Object Notation (JSON)**
 - eXtended Markup Language (XML)
 - Structured Query Language (SQL)
 - ...
- Fáceis de ler em Python!

CSV

- ficheiros de texto em que se usa vírgula para separar valores
- formato “standard” para comunicar dados tabulares

[illegible]

Excel

- O mesmo exemplo em folhas de cálculo

sfo cust sat 2014 data file_WEIGHTED_flysfo.xlsx - Excel

Samuel Mori

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do Share

M1 : X ✓ fx DEST

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	RESPNU	CCGI	RUN	INTDA	GA	ARE	STRAT	PEA	METHC	SA	AIRLIN	FLIGHT	DEST	DESTGE	DESTMAF	DEPTIM	ARRTIM	HOWL
2	1	348	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	6:30 PM	170
3	2	349	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	7:00 PM	140
4	3	350	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	8:00 PM	80
5	4	351	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:10 PM	70
6	5	352	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:20 PM	60
7	6	353	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:00 PM	140
8	7	354	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:00 PM	140
9	8	355	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:07 PM	73
10	9	356	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:00 PM	200
11	10	357	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	1:00 PM	500
12	11	358	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:50 PM	90
13	12	359	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:00 PM	200
14	13	360	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:20 PM	120
15	14	361	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:00 PM	80
16	15	362	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
17	16	363	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
18	17	364	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:30 PM	170
19	18	365	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:20 PM	120
20	19	366	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
21	20	367	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:30 PM	170

Sheet1 Sheet 1

Ready Num Lock 100%

CSV

- Ler um ficheiro CSV em Python
- E.g., índice de secura mensal para o Porto publicado pelo IPMA [aqui](#)

```
import csv
with open('mpdsi-1312-porto.csv', 'r') as f:
    table = csv.reader(f)
    data = [ row for row in table ]
print(data)
```

- Primeira linha é o cabeçalho que define o tipo de cada coluna
- Cada linha é uma lista de comprimento igual

```
['date', 'minimum', 'maximum', 'range', 'mean', 'std']
['2020-03-01', '1.17446672916', '1.3303924799',
'0.155925750732', '1.24881755738', '0.0448042640559']
```


CSV

- Converter matriz num dicionário de classificações

```
cabecalho=data[0]
meses = data[1:]
```

```
def classifica(n):
    if n>=4: return 'chuva extrema'
    elif n>=3: return 'chuva severa'
    elif n>=2: return 'chuva moderada'
    elif n>=1: return 'chuva fraca'
    elif n>-1: return 'normal'
    elif n>-2: return 'seca fraca'
    elif n>-3: return 'seca moderada'
    elif n>-4: return 'seca severa'
    else: return 'seca extrema'
```

```
meses_cs = { mes[0] : [classifica(float(n)) for n in mes[1:]] \
               for mes in meses }
```

CSV

- Selecionar a previsão mais recorrente por mês

```
def count(xs):  
    c = {}  
    for x in xs: c[x] = 1 + c.get(x, 0)  
    return c  
  
def max_count(xs):  
    c = count(xs)  
    return max(c, key=lambda k : c[k])  
  
meses_c = { mes : max_count(cs) \  
            for mes, cs in meses_cs.items() }
```

CSV

- Retornar um par com o mês mais seco de 2020 e a sua previsão

```
import dateutil.parser as date
```

```
meses_date = { date.parse(mes) : c for mes,c in meses_c.items() }  
meses_2020 = { mes.month : meses_date[mes] for mes in meses_date\  
               if mes.year == 2020 }
```

```
def desclassifica(s):  
    if s=='chuva extrema' : return 4  
    elif s=='chuva severa' : return 3  
    elif s=='chuva moderada' : return 2  
    elif s=='chuva fraca' : return 1  
    elif s=='normal' : return 0  
    elif s=='seca fraca' : return -1  
    elif s=='seca moderada' : return -2  
    elif s=='seca severa' : return -3  
    elif s=='seca extrema' : return -4  
    else : return None
```

```
def chave(mes): return desclassifica(meses_2020[mes])  
mes = min(meses_2020, key=chave)  
(mes,meses_2020[mes])
```

CSV

- Escrever num ficheiro CSV em Python
- E.g., guardar os índices de secura para de 2020

```
meses_2020_tbl = [['mes', 'secura']] \
                 + [ [m,s] for m,s in meses_2020.items() ]
```

```
with open('test.csv', 'w') as f:
    writer = csv.writer(f, delimiter=',')
    writer.writerows(meses_2020_tbl)
```


JSON

- ficheiros de texto key-value hierárquicos
- formato “standard” para troca de dados semi-estruturados entre aplicações

JSON

```
1 {
2   "sessionStart": "16-03-18-12-33-09",
3   "sessionEnd": "16-03-18-12-33-12",
4   "mapName": "TestMap",
5   "logSections": [{
6     "sector": {
7       "x": 2.0,
8       "y": -1.0,
9       "z": 0.0
10    },
11    "logLines": [{
12      "time": 37.84491729736328,
13      "state": 0,
14      "action": 1,
15      "playerPosition": {
16        "x": 24.560218811035158,
17        "y": -8.940696716308594e-8,
18        "z": 3.3498525619506838
19      },
20      "cameraRotation": {
21        "x": 0.24549755454063416,
22        "y": 0.017123013734817506,
23        "z": 0.031348951160907748,
24        "w": -0.9687389135360718
25      },
26    },
27    ...
```

XML

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <root>
3   <sessionStart>16-03-18-12-33-09</sessionStart>
4   <sessionEnd>16-03-18-12-33-12</sessionEnd>
5   <mapName>TestMap</mapName>
6   <logSections>
7     <sector>
8       <x>2</x>
9       <y>-1</y>
10      <z>0</z>
11    </sector>
12    <logLines>
13      <time>37.84491729736328</time>
14      <state>0</state>
15      <action>1</action>
16      <playerPosition>
17        <x>24.560218811035156</x>
18        <y>-8.940696716308594e-8</y>
19        <z>3.3498525619506836</z>
20      </playerPosition>
21      <cameraRotation>
22        <x>0.24549755454063416</x>
23        <y>0.017123013734817505</y>
24        <z>0.031348951160907745</z>
25        <w>-0.9687389135360718</w>
26      </cameraRotation>
27    ...
```

JSON

- Ler um ficheiro JSON em Python
- E.g., previsão metereológica de 5 dias para o Porto publicada pelo IPMA [aqui](#)
- JSON \simeq estruturas de dados Python

```
import json
with open('1131200.json', 'r') as f:
    dict = json.load(f)
print(dict)
```

```
{
  "owner": "IPMA",
  "country": "PT",
  "data": [{
    "precipitaProb": "100.0",
    "tMin": "7.5",
    "tMax": "14.5",
    ...}]
  ...
}
```

JSON

- JSON = dicionários e listas aninhados uns nos outros, cujas folhas são strings, números ou booleanos
- hierárquico: estrutura aninhada
- semi-estruturado: dicionários podem ter qualquer chave/valor; strings podem representar números, datas, etc

```
json      ::= dict
dict      ::= { string : value, ... }
value     ::= dict | sequence | basic
sequence  ::= [ value, ... ]
basic     ::= string | number | true | false | null
```

JSON

- Obter previsão para o dia mais próximo de hoje

```
import datetime
import dateutil.parser as date

data = dict['data']
weather = { date.parse(dia['forecastDate']) \
            : dia['idWeatherType'] for dia in data }

hoje = datetime.datetime.today()
dia = min(weather, key=lambda d : abs(hoje-d))
previsao = weather[dia]
```


JSON

- Converter código de previsão numa descrição textual
- Descrições fornecidas pelo IPMA [aqui](#)

```
with open('weather-type-classe.json', 'r') as f:  
    data = json.load(f) ['data']
```

```
classe = { d['idWeatherType']\  
          : d['descIdWeatherTypePT']\  
          for d in data }  
tempo = classe[previsao]
```

JSON

- Escrever num ficheiro JSON em Python
- Permite guardar grande parte dos objetos Python em ficheiro (serialização/deserialização)
- E.g., um dicionário dia : previsão textual, com uma formatação especial do dia

```
import calendar
def day_month(d):
    return str(d.day)+' '+calendar.month_abbr[d.month]
weather_dif = { day_month(d) : classe[w] for d,w in
weather.items() }
print(weather_dif)

with open("test.json","w") as f:
    json.dump(weather_dif,f)
with open("test.json","r") as f:
    weather_dif2 = json.load(f)
print(weather_dif2 == weather_dif)
```