

Programação II

+

Estruturas de Dados para Bioinformática

Formatos de dados

Hugo Pacheco

DCC/FCUP
22/23

Dados

- Nos ficheiros que temos visto até agora, 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

- Nos ficheiros que temos visto até agora, os dados estão em texto livre
- Mas têm sempre alguma estrutura, por exemplo, informação de enzimas por linha, separada por / e ‘

“REBASE version 302

staden.302

=====

*REBASE, The Restriction Enzyme Database <http://rebase.neb.com>
Copyright (c) Dr. Richard J. Roberts, 2023. All rights reserved.*

=====

Rich Roberts

Jan 27 2023

AanI/TTA'TAA//
AarI/CACCTGCNNNN'NNNN/'NNNNNNNNNGCAGGTG//
AasI/GACNNNN'NNGTC//
AatII/GACGT'C//
[...]"

Dados

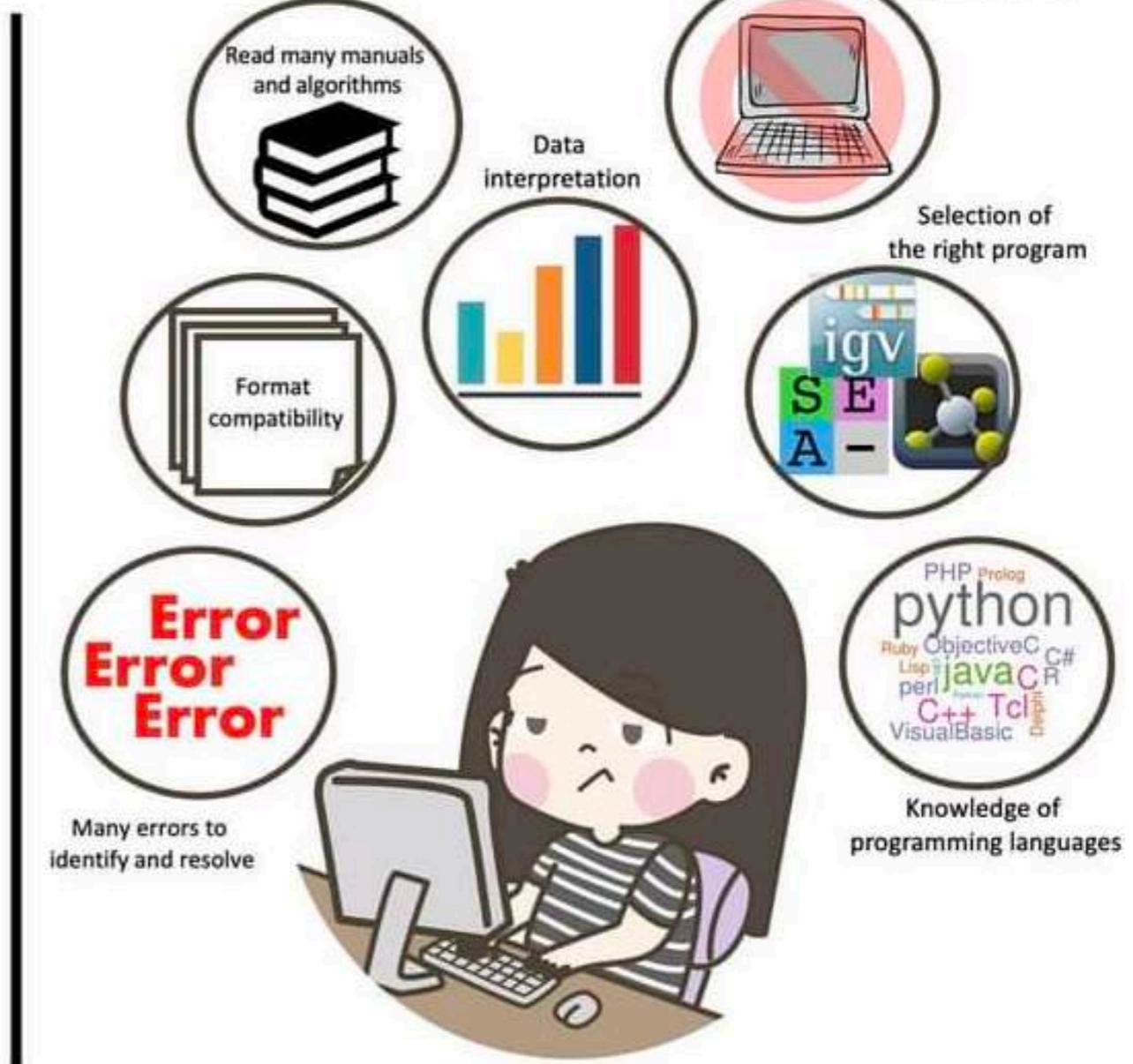
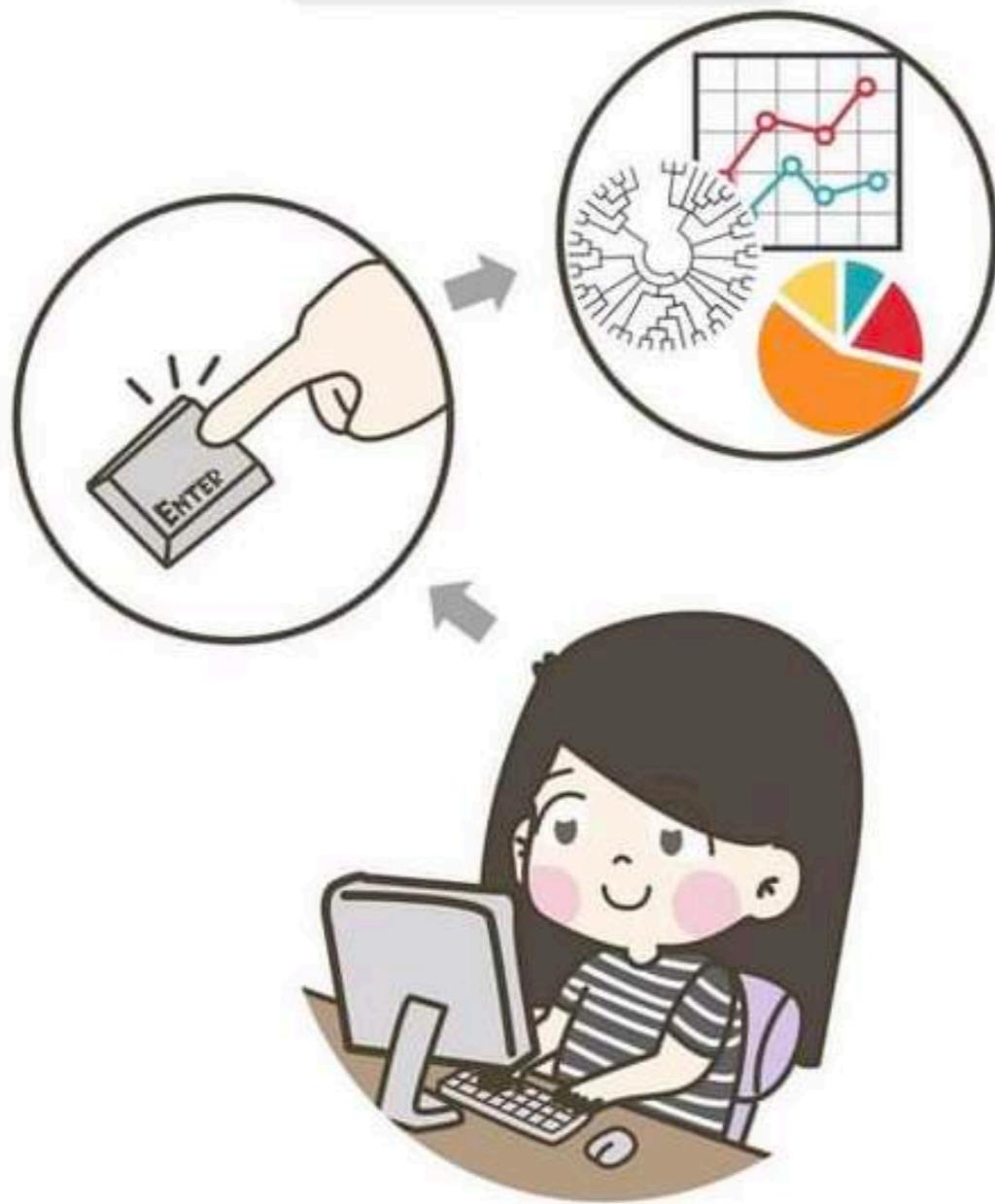
- Para ler os dados temos que explorar a sua estrutura.
- Mas se estão em texto livre...
 - pode ser difícil encontrar padrões
 - cada caso é um caso
- Idealmente, diferentes conjuntos de dados devem seguir uma estrutura comum:
 - facilitar a partilha
 - garantir preservação e suporte de longo prazo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

Dados?

Expectation

Vs

Reality



“The Web is Agreement”



[source <https://www.theweabisagreement.com/>]

Formatos de dados

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

Markdown

- ficheiros de texto em que se usa caracteres especiais para formatação
- formato “standard” para notas, blogs ou gerar HTML

```
# Projeto 1 - Análise de texto

Neste primeiro projeto vamos relembrar alguns conceitos base da programação em Python e aplicá-los no processamento de ficheiros de texto.

Aceda ao repositório [replit](https://replit.com/@up652136/Prog2-Proj1) do Projeto 1.

- Criando uma conta no [replit](https://replit.com/) e fazendo `Fork` do projeto.
- Pode consultar os ficheiros individuais na pasta [projeto1](../scripts/projeto1).
- Pode fazer download de todo o projeto como um arquivo zip [aqui](../scripts/projeto1.zip).

Neste projeto vamos processar o texto completo do *Sermão de Santo António aos Peixes do Padre António Vieira*, extrair métricas simples e reformatar o texto.

## Tarefa 1

Faça download para uma pasta local do ficheiro [sermao.txt](../scripts/projeto1/sermao.txt).

Complete a função `leTexto` que lê o conteúdo de um ficheiro de texto para uma lista de linhas.

## Tarefa 2

Analise a definição da função ``organizaSermao`` que organiza uma lista de linhas.

Por exemplo, para um sermão dado como a lista de linhas:

```python
['SERMÃO DE EXEMPLO'

```

## Projeto 1 - Análise de texto

Neste primeiro projeto vamos relembrar alguns conceitos base da programação em Python e aplicá-los no processamento de ficheiros de texto.

Aceda ao repositório [replit](#) do Projeto 1, onde pode encontrar um ficheiro `projeto1.py`:

- Criando uma conta no [replit](#) e fazendo `Fork` do projeto, pode resolver o projeto online utilizando o IDE web.
- Pode consultar os ficheiros individuais na pasta `projeto1` e fazer download dos mesmos para desenvolver o projeto no seu computador e utilizando um IDE à sua escolha.
- Pode fazer download de todo o projeto como um arquivo zip [aqui](#).

Neste projeto vamos processar o texto completo do *Sermão de Santo António aos Peixes do Padre António Vieira*, extrair métricas simples e reformatar o texto.

### Tarefa 1

Faça download para uma pasta local do ficheiro `sermao.txt`, que contém o texto integral do *Sermão de Santo António aos Peixes do Padre António Vieira*.

Complete a função `leTexto` que lê o conteúdo de um ficheiro de texto para uma lista de linhas de texto, em que cada linha de texto é uma string sem caracteres newline.

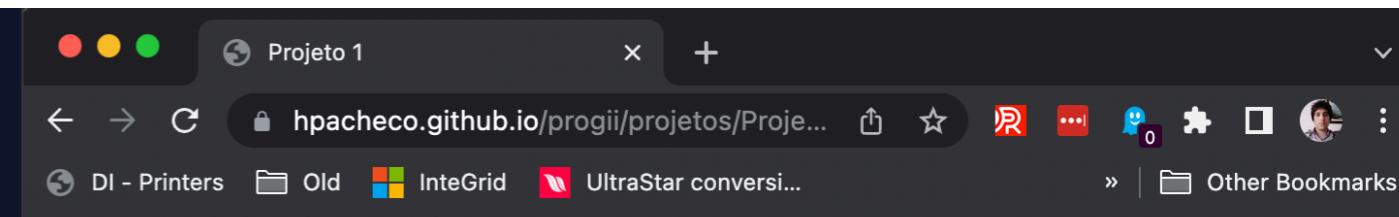
# HTML

- ficheiros de texto em que se usa a notação `<campo>texto</campo>` para definir uma estrutura hierárquica
- formato “standard” para representar páginas web

```
<body>
<header id="title-block-header">
<h1 class="title">Projeto 1</h1>
</header>
<h1 id="projeto-1---análise-de-texto">Projeto 1 - Análise de
texto</h1>
<p>Neste primeiro projeto vamos relembrar alguns conceitos base da
programação em Python e aplicá-los no processamento de ficheiros de
texto.</p>
<p>Aceda ao repositório replit do
Projeto 1, onde pode encontrar um ficheiro
<code>projeto1.py</code>:</p>

Criando uma conta no replit e
fazendo <code>Fork</code> do projeto, pode resolver o projeto online
utilizando o IDE web.
Pode consultar os ficheiros individuais na pasta projeto1 e fazer download dos mesmos
para desenvolver o projeto no seu computador e utilizando um IDE à
sua escolha.
Pode fazer download de todo o projeto como um arquivo zip aqui.

<p>Neste projeto vamos processar o texto completo do Sermão de
Santo António aos Peixes do Padre António Vieira,
extraír métricas simples e reformatar o texto.</p>
<h2 id="tarefa-1">Tarefa 1</h2>
<p>Faça download para uma pasta local do ficheiro ../scripts/projeto1/dados/sermão.txt que
```



## Projeto 1 - Análise de texto

Neste primeiro projeto vamos relembrar alguns conceitos base da programação em Python e aplicá-los no processamento de ficheiros de texto.

Aceda ao repositório replit do Projeto 1, onde pode encontrar um ficheiro projeto1.py:

- Criando uma conta no replit e fazendo Fork do projeto, pode resolver o projeto online utilizando o IDE web.
- Pode consultar os ficheiros individuais na pasta projeto1 e fazer download dos mesmos para desenvolver o projeto no seu computador e utilizando um IDE à sua escolha.
- Pode fazer download de todo o projeto como um arquivo zip aqui.

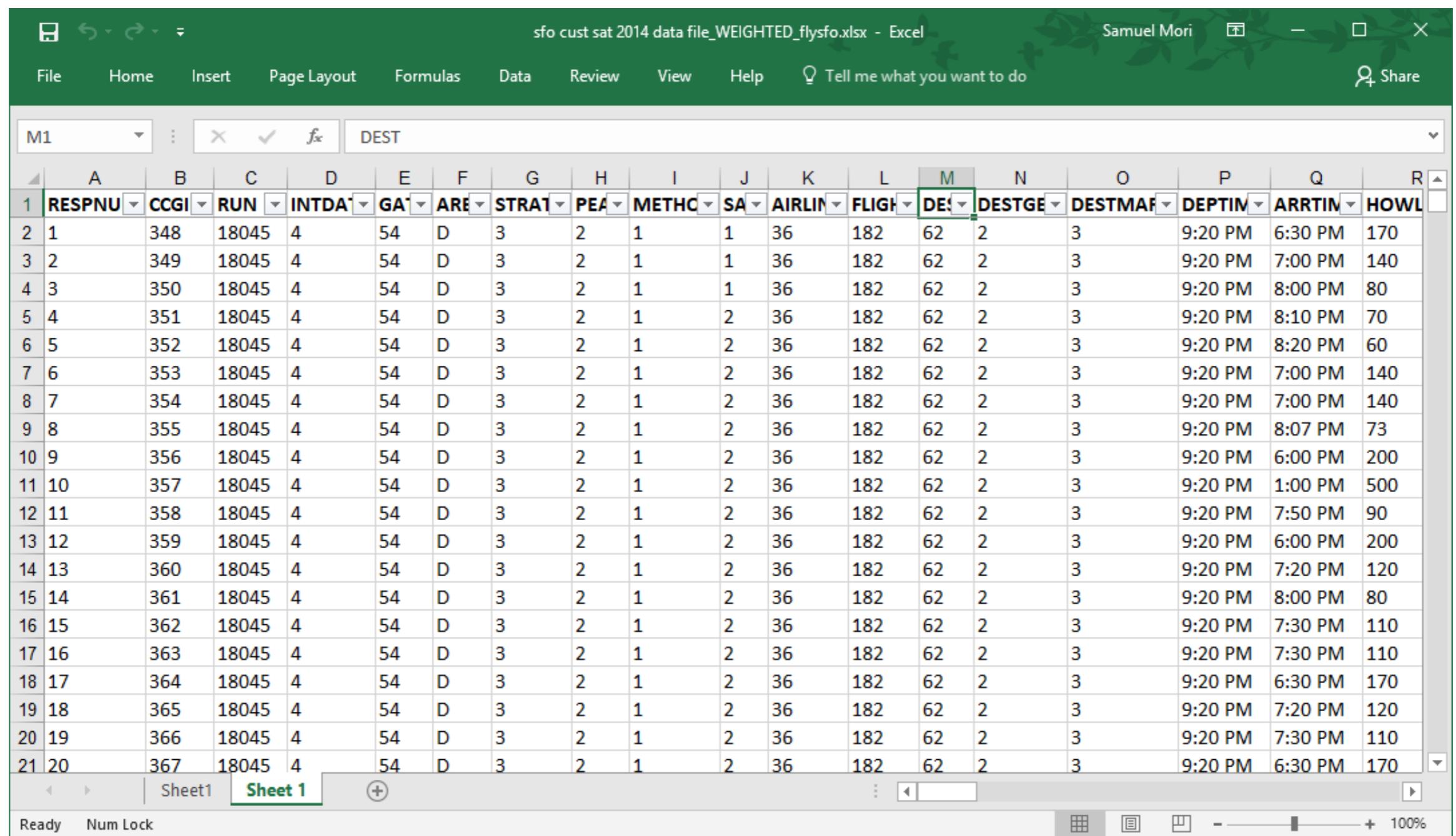
Neste projeto vamos processar o texto completo do *Sermão de Santo*

# CSV

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

# Excel

- O mesmo exemplo em folhas de cálculo



The screenshot shows a Microsoft Excel spreadsheet titled "sfo cust sat 2014 data file\_WEIGHTED\_flysfo.xlsx". The ribbon menu includes File, Home, Insert, Page Layout, Formulas, Data, Review, View, Help, and a search bar. The main area displays a data table with 21 rows and 18 columns. The columns are labeled: RESPNU, CCGI, RUN, INTDAT, GA, ARE, STRAT, PEA, METHC, SA, AIRLIN, FLIGH, DE, DESTGE, DESTMAF, DEPTIM, ARRTIN, and HOWL. Row 1 contains the column headers. The data starts from row 2, showing various flight details like route numbers, departure times, and arrival times. The cell at row 2, column M (M2) is highlighted with a green border.

RESPNU	CCGI	RUN	INTDAT	GA	ARE	STRAT	PEA	METHC	SA	AIRLIN	FLIGH	DE	DESTGE	DESTMAF	DEPTIM	ARRTIN	HOWL
1	348	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	6:30 PM	170
2	349	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	7:00 PM	140
3	350	18045	4	54	D	3	2	1	1	36	182	62	2	3	9:20 PM	8:00 PM	80
4	351	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:10 PM	70
5	352	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:20 PM	60
6	353	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:00 PM	140
7	354	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:00 PM	140
8	355	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:07 PM	73
9	356	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:00 PM	200
10	357	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	1:00 PM	500
11	358	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:50 PM	90
12	359	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:00 PM	200
13	360	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:20 PM	120
14	361	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	8:00 PM	80
15	362	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
16	363	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
17	364	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:30 PM	170
18	365	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:20 PM	120
19	366	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	7:30 PM	110
20	367	18045	4	54	D	3	2	1	2	36	182	62	2	3	9:20 PM	6:30 PM	170

# CSV

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

```
import csv
with open('.../.../dados/mpdsi-1312-porto.csv', 'r') as f:
 table = csv.reader(f)
 data = list(table)
print(data)
```

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

```
['date', 'minimum', 'maximum', 'range', 'mean', 'std']
['2022-04-01', '-2.89150762558', '-2.75911641121', '0.132391214371', '-2.83662211895', '0.0348872640209']
['2022-05-01', '-3.8392894268', '-3.70847082138', '0.130818605423', '-3.7754253887', '0.0342999110149']
```

# CSV

- Criar 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 }

```

Índice PDSI (Palmer Drought Severity Index) mensal por concelho (formato CSV)

#### Invocação:

<https://api.ipma.pt/open-data/observation/climate/mpdsi/{distrito}/mpdsi-{DICO}-{concelho}.csv>

Notas: Taxa de atualização mensal. DICO: Identificador único de concelho (de acordo com a CAOP - DGT).

- *maior ou igual a 4,0* - Chuva extrema
- *3,00 a 4,0* - Chuva severa
- *2,00 a 3,99* - Chuva moderada
- *1,00 a 1,99* - Chuva fraca
- *-0,99 a 0,99* - Normal
- *-1,99 a -1,0* - Seca fraca
- *-2,99 a -2,0* - Seca moderada
- *-3,99 a -3,0* - Seca severa
- *menor ou igual a -4,00* - Seca extrema

# 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 2022 e a sua previsão

```
import dateutil.parser as date

meses_date = { date.parse(mes) : c for mes,c in meses_c.items() }
meses_2022 = { mes.month : meses_date[mes] for mes in meses_date \
 if mes.year == 2022 }

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

mais_seco_2022 = min(meses_2022.values(),key=desclassifica)
meses_mais_secos_2022 = { k for k,v in meses_2022.items() \
 if v==mais_seco_2022 }
```

# CSV

- Escrever num ficheiro CSV em Python
- E.g., guardar os índices de secura para 2022

```
meses_2022_tbl = [['mes', 'secura']] \
 + [[m, s] for m, s in meses_2022.items()]
```

```
with open('test.csv', 'w') as f:
 writer = csv.writer(f, delimiter=',')
 writer.writerows(meses_2022_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 },
12 {"logLines": [{
13 "time": 37.84491729736328,
14 "state": 0,
15 "action": 1,
16 "playerPosition": {
17 "x": 24.560218811035158,
18 "y": -8.940696716308594e-8,
19 "z": 3.3498525619506838
20 },
21 "cameraRotation": {
22 "x": 0.24549755454063416,
23 "y": 0.017123013734817506,
24 "z": 0.031348951160907748,
25 "w": -0.9687389135360718
26 },
27 ...
28 }
```

**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, em árvore
  - 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['descWeatherTypePT'] \
 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)
```