```
1: ////////////////////////////////////////////////////
2: JSON
3: ____________________________________________________
4: maXbox Starter 82 -JSON in Code - Max Kleiner
5:
6: "There is always space for improvement"
7:     – Oscar De La Hoya
8:
9:
10: JSON (JavaScript Object Notation) is a lightweight data-interchange
    format. It is easy for humans to read and write. It is easy for machines
    to parse and generate. A JSON Parser is then used to format the JSON data
    into a properly readable JSON Format with curly brackets. That can easily
    view and identify its key and values.
11:
12:     {
13:       "date": "2021-3-4",
14:       "confirmed": 36223,
15:       "deaths": 1483,
16:       "recovered": 33632
17:     }
18:
19: Reading JSON data in maXbox could be easy. Json data can be read from a
    file or it could be a json web link. Let us first try to read the json
    from a web link.
20:
21: Const
22:     JsonUrl = 'https://pomber.github.io/covid19/timeseries.json';
23:
24: We use JSON for Delphi framework (json4delphi), it supports older versions
    of Delphi and Lazarus (6 or above) and is very versatile. Another
    advantage is the Object-pascal native code, using classes only TList,
    TStrings, TStringStream, TCollection and TStringList; The package contains
    3 units: Jsons.pas, JsonsUtilsEx.pas and a project Testunit, available at:
    https://github.com/rilyu/json4delphi
25:
26: Now we need a Load URL or Upload File function to get the json data for
    parsing. In our case load is a function-pair of open and send().
27:
28: Let us first define the necessary packages "msxml2.xmlhttp" and the JSON
    class itself:
29:
30: var  XMLhttp: OleVariant; // As Object Automation
31:      ajt: TJson; JObj: TJsonObject2;
32:
33: XMLhttp:= CreateOleObject('msxml2.xmlhttp')
34: XMLhttp.Open('GET', JsonUrl, False)   //False is async
35: XMLhttp.setRequestHeader('Content-Type','application/x-www-form-
    urlencoded');
36: XMLhttp.Send();
37:
38: response:=  XMLhttp.responseText; //assign the data
39: statuscode:=  XMLhttp.status;
40:
41: Using async = false in Open() is not always recommended, but for a few
    small requests this can be ok. Remember that the script will NOT continue
    to execute, until the server response is ready. If the server is busy or
    slow, the application will hang or stop.
42: Anyway we open our XMLhttpObject (which is late binding) as not
    asynchronous, that is to say, synchronous because we need all data to
    continue:
43:
```

```
Ref:  <class 'pandas.core.frame.DataFrame'>
RangeIndex: 76608 entries, 0 to 76607
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   country    76608 non-null  object
 1   date       76608 non-null  object
 2   confirmed  76608 non-null  int64
 3   deaths     76608 non-null  int64
 4   recovered  76608 non-null  int64
dtypes: int64(3), object(2)
memory usage: 2.9+ MB
Worldwide Covid Deaths: 2517422 at the end of Feb. 2021

The send() method (XMLhttp.Send();) needs a further explanation: send()
accepts an optional parameter which lets you specify the requests body;
this is primarily used for requests such as PUT or POST. If the request
method is GET or HEAD, the body parameter is ignored and the request body
is set to null.
Im not sure if the content-type is the right, the MIME media type for JSON
text is application/json and the default encoding is UTF-8. (Source: RFC
4627).
{content-type: application/json; charset=utf-8}
 JSON is a SUB-TYPE of text but not text alone. Json is a text
representation of an object (or array of objects). So I think both should
be allowed. The question is which works better in practice and solution.
By the way if no Accept header has been set using the setRequestHeader(),
an Accept header with the type "*/*" (any type) is sent.


Next we define the Json instance:

ajt:= TJson.create();

For slicing (filter) the data we copy the range from response
timeseries.json:

startR:= pos('"'+ACOUNTRY1+'"',response);
stopR:= pos('"'+ACOUNTRY2+'"',response);
writeln('DataLen Overall: '+itoa(length(response)))
resrange:= Copy(response, startR, stopR-startR);

Now we parse the response.

  try
    ajt.parse(resrange);
  except
    writeln( 'Exception: <TJson>"" parse error: {'+
                  exceptiontostring(exceptiontype, exceptionparam))
  end;

Now we can iterate through the keys with values as items. Here, in the
above sample JSON data: date, confirmed, deaths and recovered are known as
key and "2020-1-22", 0, 0 and 0 known as a Value. All Data are available
in a Key and value pair.

First we get a list of all 192 country names as the node name:

JObj:= ajt.JsonObject;
  writeln('Get all Countries: ')
  for cnt:= 0 to jobj.count-1 do
     writeln(Jobj.items[cnt].name);
```

```
     ...United Kingdom
        Uruguay
        Uzbekistan
        Vanuatu
        Venezuela
        Vietnam...

So the country is an object to get. Ok, it is a JsonObject dictionary with
192 countries. Lets check the keys of our dict with a nested loop of all
confirmed cases:

  for cnt:= 0 to Jobj.count-1 do begin
    Clabel:= Jobj.items[cnt].name;
    JArray2:= jobj.values[Clabel].asArray;
    for cnt2:= 0 to jarray2.count-1 do
      itmp:= jarray2.items[cnt2].asObject.values['confirmed'].asinteger;
  end;

So we pass the country name items[cnt].name from the Json Object in to the
value list and we get back an a object array for the second iteration of
at the moment 408 records with key values of integer format. Our dimension
for now is 192 * 408 = 78336 records.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78336 entries, 0 to 78335
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   country    78336 non-null  object
 1   date       78336 non-null  object
 2   confirmed  78336 non-null  int64
 3   deaths     78336 non-null  int64
 4   recovered  78336 non-null  int64
dtypes: int64(3), object(2)
memory usage: 3.0+ MB
  {
  "Afghanistan": [
    {
      "date": "2020-1-22",
      "confirmed": 0,
      "deaths": 0,
      "recovered": 0
    },
    {
      "date": "2020-1-23",
      "confirmed": 0,
      "deaths": 0,
      "recovered": 0
    },...

In a second attempt we visualize the timeseries with TeeChart Standard. Ok
we got the objectarray as sort of dataframe with items and values but not
in the form that we wanted. We will have to unwind the nested data like
above to build a proper dataframe with chart series at runtime for TChart:

 Chart1.Title.Text.clear;
 //AssignSeries(OldSeries,NewSeries:TChartSeries);
 Chart1.Title.Text.add('Sciplot Serie: '+'World Covid21 confirmed not †');

 Chart1.Axes.Bottom.Title.Caption:= 'Days from  '+
                 datetimetostr(date-400)+' to '+datetimetostr(date-1);
```

```
146:  Chart1.BottomAxis.Labels:= True;
147:  Chart1.LeftAxis.Logarithmic:= true;
148:  //Chart1.XValues.Multiplier:= 1
149:  Clabel:='';
150:    for cnt:= 0 to Jobj.count-1 do begin
151:      Clabel:= Jobj.items[cnt].name
152:      JArray2:= jobj.values[Clabel].asarray;
153:      chart1.AddSeries(TFastLineSeries.Create(Self)); //runtime instances
154:      chart1.series[cnt].title:= Clabel;
155:      TFastLineSeries(chart1.series[cnt]).LinePen.Width:= 4;
156:      for cnt2:= jarray2.count-400 to jarray2.count-1 do begin
157:        itmp:= jarray2.items[cnt2].asObject.values['confirmed'].asinteger;
158:        sumup:= sumup+ itmp
159:        chart1.Series[cnt].Addxy(cnt2,itmp, itoa(cnt2),clRed);
160:      end;
161:    end;
162:  writeln('Worldwide Count:'+itoa(ajt.count)+' Covid Confirm:
      '+itoa(sumup));//*)
163:
164: The plot you can find at:
165:
166: http://www.softwareschule.ch/examples/covid3.png
167:
168: TeeChart is a charting library for programmers, developed and managed by
     Steema Software of Girona, Catalonia, Spain. It is available as commercial
     and non-commercial software. TeeChart has been included in most Delphi and
     C++Builder products since 1997, and TeeChart Standard currently is part of
     Embarcadero RAD Studio 10.4 Sydney.
169:
170:
171: Conclusion:
172: The proper way to use JSON is to specify types that must be compatible at
     runtime in order for your code to work correctly.
173:  The TJsonBase= class(TObject) and TJsonValue= class(TJsonBase) namespace
     contains all the entry points and the main types. The TJson=
     class(TJsonBase) namespace contains attributes and APIs for advanced
     scenarios and customization.
174:  Those are the supported types:
175: type
176:   TJsonValueType =
177:             (jvNone,jvNull,jvString,jvNumber,jvBoolean,jvObject,jvArray);
178:   TJsonStructType = (jsNone, jsArray, jsObject);
179:   TJsonNull = (null);
180:   TJsonEmpty = (empty);
181:
182: https://github.com/rilyu/json4delphi/blob/master/src/Jsons.pas
183:
184: JSON Parser Tools should have the following main functionality:
185: •  Directly New your Record.
186: •  Sava Temporary JSON Data.
187: •  Copy and Paste JSON Data.
188: •  Download JSON Data.
189: •  Share Temporary Data anywhere.
190: •  Load JSON URL directly into Editor.
191: •  Redo and Undo facility when you edit your JSON online.
192: •  JSON Validator for your Online Changes and your other JSON Data.
193: •  Minify or Compact JSON Data to resave and reduct its Size.
194: •  In Treeview, You can Search and highlight, and Sorting Data.
195: From: https://jsonparser.org/
196:
197: Appendix for Python to get JSON Data:
198:
```

```
199: >>> import requests
200: >>> import pandas as pd
201: >>> data = requests.get('https://pomber.github.io/covid19/timeseries.json')
202: >>> jsondata = data.json()
203: >>> df = pd.DataFrame.from_dict(jsondata)
204: >>> df.info()
205: <class 'pandas.core.frame.DataFrame'>
206: RangeIndex: 408 entries, 0 to 407
207: Columns: 192 entries, Afghanistan to Zimbabwe
208: dtypes: object(192)
209: memory usage: 612.1+ KB
210:
211: Ref:
212:     http://www.softwareschule.ch/examples/covid2.txt
213:     https://github.com/rilyu/json4delphi
214:     https://pomber.github.io/covid19/timeseries.json
215:     http://www.softwareschule.ch/examples/unittests.txt
216:     script: 1026_json_automation_refactor2.txt
217: Doc:
218:     https://maxbox4.wordpress.com
219: >>> https://my6.code.blog/2021/03/03/json-automation/
220: >>> https://entwickler-konferenz.de/speaker/max-kleiner/
```