Function:

A python function has been created in order to access the *SPECS-EUPORIAS Data Portal* in a user-friendly way, allowing the retrieval of dimensional slices of selected simulation members from the ECMWF's SYSTEM4 model. This function (?load_system4.py) automatically cares about the proper location of the right indices for data sub-setting across the different variable dimensions, given a few simple arguments for subset definition. In addition, instead of retrieving a NetCDF file that needs to be opened and read, the requested data is directly loaded into the current python working session, according to a particular structure described below, prior to data analysis and/or representation.

The request is simply formulated via the `load_system4` function:

```
>>> load_system4(dataset, var, season, leadMonth, lonLim, latLim, year, members=[])
```

The arguments of the function are described below:

- `dataset`: A character string indicating the full URL path to the OPeNDAP dataset. Currently, the accepted values correspond to the System4 datasets described in Section ?Datasets, for instance `http://www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Members.ncml`, but using the `System4_Seasonal_15Members.ncml`, `System4_Seasonal_51Members.ncml` or `System4_Annual_15Members.ncml` ending strings depending on the dataset of choice.
- `var`: Variable code. Argument values currently accepted are `tas`, `tasmin`, `tasmax`, `pr` or `mslp`, as internally defined in the vocabulary for System4 following the nomenclature displayed in the table below. However, note that new variables and datasets will be progressively included and that depending on the time step of the variable the units might be referred to different time aggregations. For instance, currently `mslp` is 6-hourly, and thus the 6-hourly mean value is returned for each time step. Similarly, 24-h accumulated values are returned for `pr`, and so on. Note that the *instantaneous* and *aggregated* fields in table below refer to the potential time step values that the variables may take, which does not mean that the resolution provided by the System4 model is necessarily that.

Short Name	Long name	Units	Instantaneous	Aggregated
tasmax	Maximum temperature at 2 metres	K	No	Yes
tasmin	Minimum temperature at 2 metres	K	No	Yes
tas	Mean temperature at 2 metres	K	Yes	Yes
pr	Total precipitation accumulated	mm	No	Yes
mslp	Mean sea level pressure	Pa	Yes	Yes

- `members`: List of members to select. In the above case, a single member (the first) of the System4 ensemble is loaded, but additional members could be also specified (e.g. `members=[0,1,2,3,4]` for the first five members).
- `lonLim`: Vector of length = 2, with minimum and maximum longitude coordinates, in decimal degrees, of the bounding box selected.
- `latLim`: Vector of length = 2, with minimum and maximum latitude coordinates, in decimal degrees, of the bounding box selected.
- `season`: A vector of integers specifying the desired season (in months, January=1, etc.) of analysis. Options include a single month (as in the above example) or a standard season (e.g. `season = [12,1,2]` for standard Boreal winter, DJF).
- `year`: List of years to select. Note that in cases with year-crossing seasons (e.g. winter DJF, `season = [12,1,2]`, for a particular year period `year = [1981,1982,1983]`), by convention the first season would be DJF 1981/82.
- `leadMonth`: Lead month forecast time corresponding to the first month of the specified season. Note that `leadMonth = 1` for `season = [1]` (January) corresponds to the December initialization forecasts. In this way the effect of the lead time forecast in the analysis of a particular season can be analyzed by just changing this parameter.

The output returned by the function consists of a list of user data objects (one for each member loaded) with the following methods that provide the necessary information for data representation and analysis:

- `ud.short_name`: Character string indicating the variable short name, as defined in the vocabulary (see Table above).
- `ud.units`: Unicode text. Units of the variable.
- `ud.times`: Array of datetime objects. It indicates the time span of each forecast time.
- `ud.member`: List of length *n*, where *n* is the number of members of the ensemble selected by the `members` argument.

- `ud.LatLonCoords`: A 2-D matrix of *j* columns (where *j* = number of grid points selected) and two rows corresponding to the longitude and latitude coordinates respectively.
- `ud.runtime`: Array of datetime objects corresponding to the initialization times selected.
- `ud.data`: A 2-D matrix of *i* rows and *j* columns, *i* represents the forecast times and *j* the grid-points selected.

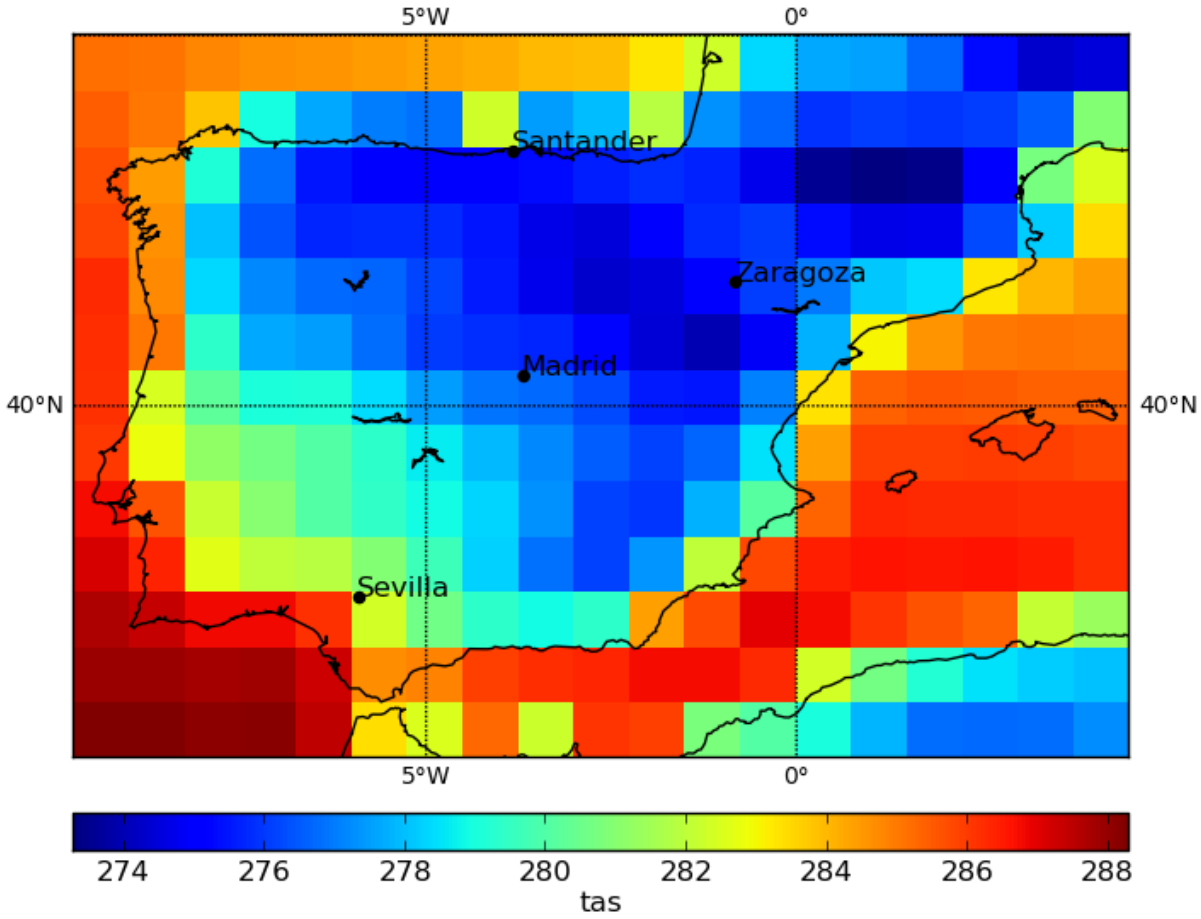
Example:

An illustrative example of the `load_system4` function is described in the next lines. We will retrieve System4 simulation data for the Iberian Peninsula, considering mean surface temperature for January and the first simulation member, for the 10-year period 1990-1999. It should be noted that the user must enter here his/her authorized username and password as character strings.

```
>>> var = "tas"
>>> season = [1]
>>> leadMonth = 1
>>> lonlim = [-10,5]
>>> latlim= [35,45]
>>> year = [1990,1991,1992,1993,1994,1995,1996,1997,1998,1999]
>>> members = [0]
>>> username = "myUsername"
>>> password = "myPassword"
>>> dataset = "http://%s:%s@www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Members.ncml" %(username,password)
>>> ud = loadSystem4(dataset,var,season, leadMonth,lonlim, latlim, year, members=[0])
```

Data is now loaded into the python session. One of the most common tasks consists on the representation of data, e.g. by mapping the spatial mean of the period under consideration. In addition, we will also add to the map the point locations of four Spanish cities. It can be done easily:

```
>>> cities_name=np.array(["Sevilla","Madrid","Santander","Zaragoza"])
>>> cities_latlon=np.array([[(37.41),(-5.9)],[(40.40),(-3.68)],[(43.43),(-3.82)],[(41.67),(-0.82)]])
>>> temporal_mean=ud.data.mean(axis=0)
>>> plot_map(temporal_mean,ud,season,var,cities_latlon,cities_name,method="pixels")
```



Another common task is the representation of time series for selected point locations/grid cells. In this example, we will display time series of the requested dataset at the four grid points coincident with the cities represented in the map. To this aim, we will create a function that search the data of the nearest grid points to the specified locations.

```
>>> def plot_serie_cities(cities_latlon,cities_name,ud):
...     pcolors = {
...       "0":"blue",
...       "1":"green",
...       "2":"red",
...       "3":"purple",
...       "4":"cyan",
...       "5":"yellow",
...       "6":"magenta",
...       "7":"pink",
...       "8":"orange",
...       "9":"brown",
...       "10":"grey",
...     }
...     fig=plt.figure()
...     ax = fig.add_subplot(111)
...     x=np.arange(len(ud.times))
...     xticks_mask = [d.month in season and d.day == 1 for d in ud.times]
...     xticks = x[np.array(xticks_mask)]
...     ax.set_xticks(xticks)
...     ticklabels=[t.strftime("%Y-%m") for t in ud.times[np.array(xticks_mask)]]
...     ax.set_xticklabels(ticklabels)
```

```
...     plt.xticks(rotation=25)
...     plt.ylabel("%s (%s)" %(var,ud.units))
...     lat = cities_latlon[:,0]
...     lon = cities_latlon[:,1]
...     for i in np.arange(len(cities_name)):
...        city_position=(((ud.LatLonCoords[0,:]-lon[i])**2)+((ud.LatLonCoords[1,:]-lat[i])**2)).argmin()
...        city = ud.data[:,city_position]
...        c=("%("+str(i)+")s") % pcolors
...        plt.plot(city,color=c, label=cities_name[i])
...     plt.legend(loc="lower left",prop={'size':'small'})
...     plt.savefig("map_serie_all_cities.png")
```

Once we hace the function defined, we use it to plot the graph.

```
>>> plot_serie_cities(cities_latlon,cities_name,ud)
```