Part 4 - Useful packages for Librarians

importing packages

Often times you need to import some packages that contain functions to help you do things more quickly, and often times you give that package a nickname like so:

```
import package_name as package_nickname
```

therefore you can use `package_nickname` later in your code to access the packages functionality.

Pandas, a package for working with tabular data

Pandas is one of the most popular packages in python, as it works with spreadsheet style data. You can quickly import a csv with its import function:

```
import pandas as pd # everyone uses pd as the nickname for pandas

df = pd.read_csv('data/dataset.csv')
```

When reading or writing files in Python, by default, the jupyter notebook will look in the folder where the notebook is saved, therefore in the example above, we are using a relative subfolder called `data` with a `dataset.csv` saved in it.

Accessing data with pandas;

The beauty of Python is that all packages try to work in the same way as the base kinds of objects in python, so a Pandas dataframe, which is the way Pandas represents a table, can be thought of as a dictionary full of lists, where the dictionary key is the name of the column, and the list index is the row number. Here is an example:

```
df['column 1'][0] # will output the value of the first row in column 1
```

Handy things to do with a dataframe:

```
df.dtypes #an attribute that contains the type of data in each column
df.head() # a method that prints the top five rows of the table
df.shape # an attribute tells you the number of rows and columns
df.columns # an attribute that tells you the names of all the columns
```

Note: Dataframes also have attributes, which are kind of like methods but without the `()` You can think of them as metadata variables attached to the dataframe, and accessed using the full stop and the name of the attribute.

Another super handy package, dcxml

This package was created to make dublin core xml files in Python, all we need to do is first import the package:

```
from dcxml import simpledc
```

Here we are using a slightly different code, where we are importing one thing (simpledc) from the package (dcxml). simpledc is a special python object that carries several methods to create dublin core xml.

To create a string xml file, we will use the `simpledc.tostring()` method, which takes in data in a dictionary format:

```
data = {
        'contributors' : ['CERN'],
        'coverage' : ['Geneva'],
        'creators' : ['CERN'],
        'dates' : ['2002'],
        'descriptions' : ['Simple Dublin Core generation'],
        'formats' : ['application/xml'],
        'identifiers' : ['dublin-core'],
        'languages' : ['en'],
        'publishers' : ['CERN'],
        'relations' : ['Invenio Software'],
        'rights' : ['MIT'],
        'sources' : ['Python'],
        'subject' : ['XML'],
        'titles' : ['Dublin Core XML'],
        'types' : ['Software'],
         'extra': ['extra']
       }
```

All we need to do is put that dictionary into the `simpledc.tostring()` method and we have an xml!

```
xml = simpledc.tostring(data) # takes in a dictionary
print(xml) # print it out and see!
```

Magic doodad, convert_df_row_to_dictionary

This doodad was created for this workshop, and will help us convert one of our metadata records, as a row of a dataframe, into a dictionary full of lists, to prepare it for the `simpledc.tostring()` method:

this function takes in a dataframe as the first argument, and a row index number as the second, and outputs a dictionary formatted for the simpledc methods:

```
from jgarber_respitch.workshop_tools import convert_df_row_to_dictionary

convert_df_row_to_dictionary(df,1)
```