How to Create Lists in SAS and Python – Code Examples

Create a horizontal list of values from a variable

- Create Macro variables using PROC SQL Into clause in SAS
- Convert unique values to a Python List in Python

Create multiple lists from various variables

- SAS Macro (multiple ways) vs. Python Function

Data

The Sashelp.demographics data set provides the 2004 revision of data derived from world population prospects. The data set contains 197 observations and 18 variables, including the following three (i.e., NAME, pop, and region). Below are the attributes of those three variables.

```
NAME        TYPE              LABEL

NAME       Character     GLC Country Name
pop        Numeric       Population (2005)
region     Character     Region
```

Task 1: Create a list of country names for region="AFR" using SAS

```
1     dm "log; clear; output; clear; odsresults; clear;";
2     proc sql noprint;
3         select distinct trim(name)
4             INTO :AFR_clist separated by ','
5              FROM SASHELP.demographics
6              where region="AFR";
7     quit;
8
9  %put NOTE: Processing region AFR (Nonmacro code);
10 %put Number of countries for AFR_clist = &sqlobs;
11 %put &=AFR_clist;
12 %put NOTE: End of region AFR processing;
```

Line 1 uses the DM statement to clear the log, output, and ODS results windows. This one-liner is often used to clean up the SAS environment before running new code.

Line 2 initiates an SQL procedure in SAS. The noprint option suppresses the automatic printing of results to the output window.

Line 3 selects unique (distinct) values of the 'name' column, with any leading or trailing spaces removed (trim).

Line 4 creates a macro variable named AFR_clist. The selected values will be stored in this variable, separated by commas.

Line 5 specifies the dataset to query. In this case, it's the 'demographics' dataset from the SASHELP library.

Line 6 is the WHERE clause that filters the data. It only selects rows where the 'region' column equals "AFR" (likely standing for Africa).

Line 7 ends the SQL procedure.

In summary, this code creates a macro variable (:AFR_clist) containing a comma-separated list of all unique country names in the Africa region from the SASHELP.demographics dataset. This list could be used later in your SAS program for further processing or analysis of African countries.

Task 2: Create a list of country names for region="AFR" using Python

```
1 | import pandas as pd
2 |
3 | # Load the dataset
4 | demographics = pd.read_csv('c://Data//sashelp_demographics.csv')
5 |
6 | # Select distinct names for the AFR region
7 | afr_clist = demographics[demographics['region'] ==
'AFR']['name'].unique().tolist()
8 |
9 | # Convert list to a comma-separated string
10| afr_clist_str = ','.join(afr_clist)
11|
12| # Output the number of countries and the list
13| print(f"Number of countries for AFR_clist = {len(afr_clist)}")
14| print(f"AFR_clist = {afr_clist_str}")
15|
```

Line 1 imports the pandas library and aliases it as 'pd', a standard convention.

Line 3 reads a CSV file from the specified path and creates a pandas DataFrame called 'demographics'.

Line 7 does several things:

- `demographics['region'] == 'AFR'` creates a boolean mask for rows where the 'region' column is 'AFR'.
- demographics[demographics['region'] == 'AFR'] selects only those rows.
- `['name']` selects only the 'name' column from those rows.
- `.unique()` gets the unique values from that column.
- `.tolist()` converts the unique values to a Python list.
- `','.join(afr_clist)` Converts the list to a comma-separated string.

Line 10 joins all elements in the afr_clist into a single string, separating them with ', '.

Line 13 prints the number of countries (using len(afr_clist)) and the comma-separated list of country names.

Line 14 effectively filters the dataset for countries in the AFR region, creates a list of unique country names, and then outputs both the count and the names of these countries.

Task 3: Create a list of country names for each of the six regions using SAS – Multiple ways

```
1 | * Macro approach 1;
2 | %macro runit(rname);
3 | proc sql noprint;
4 |     select distinct trim(name)
5 |         INTO :&rname._clist separated by ','
6 |         FROM SASHELP.demographics
7 |         where region="&rname";
8 | quit;
9 | %put NOTE: Processing region &rname using a macro with positional parameter;
10| %put &rname._clist = %superq(&rname._clist);
11| %put Number of countries for &rname._clist = &sqlobs;
12| %put NOTE: End of region &rname processing;
13| %mend runit;
14| %runit(AFR)
15| %runit(AMR)
16| %runit(EMR)
17| %runit(SEAR)
18| %runit(WPR)
19| %runit(EUR)
20|
21| * Macro approach 2;
22| %macro process_region(rname);
23|     proc sql noprint;
24|         select distinct trim(name)
25|             into :&rname._clist separated by ","
26|             from SASHELP.demographics
27|             where region="&rname";
28|     quit;
29|     %put NOTE: Processing region &rname using a macro Loop;
30|     %put NOTE: SQL Observation count = &sqlobs;
31|     %put &rname._clist = %superq(&rname._clist);
32|     %put NOTE: End of region &rname processing;
33| %mend process_region;
34| %macro run_all;
35|     %let regions = AFR AMR EMR SEAR WPR EUR;
36|     %do i = 1 %to %sysfunc(countw(&regions));
37|         %let region = %scan(&regions, &i);
38|         %process_region(&region)
39|     %end;
40| %mend run_all;
41| %run_all
42|
43| * Macro approach 3;
44| %macro process_region(rname);
45|     proc sql noprint;
46|         select distinct trim(name)
47|             into :&rname._clist separated by ","
48|             from SASHELP.demographics
49|             where region="&rname";
50|     quit;
51|     %put NOTE: Processing region &rname using macro with CALL EXECUTE;
52|     %put NOTE: SQL Observation count = &sqlobs;
53|     %put &rname._clist = %superq(&rname._clist);
54|     %put NOTE: End of region &rname processing;
55| %mend process_region; 56|
data _null_;
57|     length region $4;
58|     array regions[6] $4 ('AFR' 'AMR' 'EMR' 'SEAR' 'WPR' 'EUR');
59|     do i = 1 to dim(regions);
60|         region = regions[i];
61|         call execute('%nrstr(%process_region(' || region || '));');
62|     end;
63| run;
```

Task 4: Create a list of country names for each of the six regions using Python

```python
import pandas as pd

def runit(rname):
    # Load the dataset
    demographics = pd.read_csv('c:\\Data\sashelp_demographics.csv')  # Assuming the data is in a CSV format

    # Select distinct names based on the region
    clist = demographics[demographics['region'] == rname]['name'].unique()
    clist_str = ', '.join(clist)

    print("Providing values to the function call")
    print(f"NOTE: Processing region {rname}")
    print(f"{rname}_clist = {clist_str}")
    print(f"Number of countries for {rname}_clist = {len(clist)}")
    print(f"NOTE: End of region {rname} processing")
    print()

# Call the function for each region
for region in ['AFR', 'AMR', 'EMR', 'SEAR', 'WPR', 'EUR']:
    runit(region)
```

This code defines a function runit that processes demographic data for different regions and prints information about the countries in each region. Here's a breakdown of what the code does:

1. It imports the pandas library as pd.
2. The runit function is defined, which takes a region name (rname) as an argument.
3. Inside the function:
    - It loads a CSV file named 'sashelp_demographics.csv' from the 'C:\Data' directory into a pandas DataFrame.
    - It filters the DataFrame for the given region and selects unique country names.
    - It prints information about the selected countries, including:
        - A note about processing the region
        - A list of countries in that region
        - The number of countries in the region
        - A note indicating the end of processing for that region
4. After the function definition, a loop calls runit for each region: 'AFR', 'AMR', 'EMR', 'SEAR', 'WPR', and 'EUR'.

This code will process and print information for each region using the CSV file's data.