4. Desarrollo de aplicaciones con uso de listas y tablas basadas en Numpy y Pandas

4.1 Biblioteca Numpy.

Las listas son arreglos o *arrays* unidimensionales que permiten almacenar más de un elemento de datos similares o diferentes accesibles por su posición referenciado a un índice. En *Python*, una lista almacena a los elementos encerrados entre corchetes y separados por comas. Por ejemplo, *edad=[30,45,55,30,52,40]*, o *apellido=["López","Pérez","Martínez","González"]*. Las listas inician con el índice 0; por ejemplo, *edad[2]=55* o *nombre[0]="López"*.

Las tablas son arreglos o *arrays* bidimensionales formados por listas de listas de la misma longitud; por ejemplo, *identidad=[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]*. Un elemento de esta tabla se accede de la siguiente manera: *identidad[1][1]=1*. Recordar que los índices inician en 0.

En *Numpy*, los elementos de una lista no están separadas por comas, pero inician en el índice 0 y se acceden de forma similar a las listas originales de *Python*, aunque existen algunas variantes interesantes. En el siguiente programa, se muestra la forma de operar los arreglos (listas y tablas) en *Numpy*.

```
import numpy as np
import numpy.linalg as al
lista1=list(range(10))
lista2=np.arange(10)
print("Lista1: ",lista1)
print("Lista2: ",lista2)
listaaleatoria=np.random.rand(5)
print("Lista aleatoria: ")
print(listaaleatoria)
tablaaleatoria=np.random.rand(3,3)
print("Tabla Aleatoria: ")
print(tablaaleatoria)
print("Tabla Aleatoria * 10:")
print(tablaaleatoria*10)
print("Tabla Aleatoria+TablaAleatoria:")
print(tablaaleatoria+tablaaleatoria)
arreglo1=np.array(lista1)
print("Arreglo 1:")
print(arreglo1)
arreglo2=np.array(listaaleatoria)
print("Arreglo 2:")
print(arreglo2)
arreglo3=np.array([10,20,30,40,50])
print("Arreglo 3:")
print(arreglo3)
lista3=[listaaleatoria,listaaleatoria*5]
print("Lista 3:")
print(lista3)
arreglo4=np.array(lista3)
print("Arreglo 4:")
print(arreglo4)
```

```
lista0=np.zeros(5)
print("Lista 0:")
print(lista0)
matriz0=np.zeros((3,3))
print("Matriz 0:")
print(matriz0)
listaunos=np.ones(5)
print("Lista unos:")
print(listaunos)
tablaunos=np.ones((3,3))
print("Matriz unos")
print(tablaunos)
arr16=np.array(np.random.rand(4,4),dtype=np.float16)
print("Arreglo 16 bits:")
print(arr16)
arr32=np.array(np.random.rand(4,4),dtype=np.float32)
print("Arreglo 32 bits:")
print(arr32)
arr64=np.array(np.random.rand(4,4),dtype=np.float64)
print("Arreglo 64 bits:")
print(arr64)
print("Transpuesta de una matriz:")
print(arr32.T)
print("Otra instrucción para la transpuesta")
print(arr64.transpose())
print("Multiplicación elemento a elemento de dos matrices:")
print(arr64*arr32)
print("Multiplicación de dos matrices A(3x3)*B(3x2)")
arrA=np.array([[2,3,4],[4,5,7],[3,2,1]])
arrB=np.array([[1,5],[4,3],[7,9]])
print("Matriz A(3x3):")
print(arrA)
print("Matriz B(3x2):")
print(arrB)
print("Matriz C=A*B(3x2)")
arrC=np.dot(arrA,arrB)
print(arrC)
print("Matriz inversa de A:")
arrAInv=al.inv(arrA)
print(arrAInv)
print("Matriz identidad I=A@A^(-1)")
print("A@B=dot(A,B)")
arrI=arrA@arrAInv
print(arrI)
arrQ,arrR=al.qr(arrA)
print("Matriz ortogonal Q:")
print(arrQ)
print("Matriz triangular superior R:")
print(arrR)
print("Matriz A=Q@R:")
print(np.dot(arrQ,arrR))
print("Determinante de A: ",al.det(arrA))
print("Se verifica det(A)=Prod(r[i,i]):")
producto=1
for i in range(0,len(arrA)):
    producto=producto*arrR[i,i]
print("Determinante de A es:",producto)
```

```
Lista1:  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Lista2:  [0 1 2 3 4 5 6 7 8 9]
Lista aleatoria:
[7.68762176e-01 8.73388317e-01 2.16416333e-01 3.60097675e-04
 6.85303158e-02]
Tabla Aleatoria:
[[0.9574978  0.31881483 0.66065693]
 [0.26727239 0.85451391 0.54388205]
 [0.53556509 0.44177202 0.6589062 ]]
Tabla Aleatoria * 10:
[[9.57497798 3.18814827 6.60656931]
 [2.67272386 8.54513915 5.4388205 ]
 [5.35565093 4.41772024 6.58906198]]
Tabla Aleatoria+TablaAleatoria:
[[1.9149956  0.63762965 1.32131386]
 [0.53454477 1.70902783 1.0877641 ]
 [1.07113019 0.88354405 1.3178124 ]]
Arreglo 1:
[0 1 2 3 4 5 6 7 8 9]
Arreglo 2:
[7.68762176e-01 8.73388317e-01 2.16416333e-01 3.60097675e-04
 6.85303158e-02]
Arreglo 3:
[10 20 30 40 50]
Lista 3:
[array([7.68762176e-01, 8.73388317e-01, 2.16416333e-01, 3.60097675e-04,
       6.85303158e-02]), array([3.84381088e+00, 4.36694158e+00, 1.08208167e+00,
1.80048838e-03,
       3.42651579e-01])]
Arreglo 4:
[[7.68762176e-01 8.73388317e-01 2.16416333e-01 3.60097675e-04
  6.85303158e-02]
 [3.84381088e+00 4.36694158e+00 1.08208167e+00 1.80048838e-03
  3.42651579e-01]]
```

```
Lista 0:
[0. 0. 0. 0. 0.]
Matriz 0:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Lista unos:
[1. 1. 1. 1. 1.]
Matriz unos
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Arreglo 16 bits:
[[0.0953   0.3455   0.003008 0.3696  ]
 [0.6885   0.01846  0.4695   0.419   ]
 [0.905    0.9214   0.2307   0.8135  ]
 [0.3774   0.0328   0.956    0.533   ]]
Arreglo 32 bits:
[[0.9186976  0.48057666 0.09981839 0.5460421 ]
 [0.68884164 0.7543699  0.52816033 0.74343026]
 [0.6260621  0.79499906 0.5206309  0.23532014]
 [0.04095504 0.00462493 0.31121805 0.86380845]]
Arreglo 64 bits:
[[0.64924251 0.42495742 0.94303141 0.46864466]
 [0.77399023 0.72412562 0.08530071 0.20370253]
 [0.91901318 0.74342124 0.60477553 0.84364215]
 [0.95128822 0.61161741 0.1795824  0.9445369 ]]
Transpuesta de una matriz:
[[0.9186976  0.68884164 0.6260621  0.04095504]
 [0.48057666 0.7543699  0.79499906 0.00462493]
 [0.09981839 0.52816033 0.5206309  0.31121805]
 [0.5460421  0.74343026 0.23532014 0.86380845]]
Otra instrucción para la transpuesta
[[0.64924251 0.77399023 0.91901318 0.95128822]
 [0.42495742 0.72412562 0.74342124 0.61161741]
 [0.94303141 0.08530071 0.60477553 0.1795824 ]
 [0.46864466 0.20370253 0.84364215 0.9445369 ]]
```

```
Multiplicación elemento a elemento de dos matrices:
[[0.59645753 0.20422462 0.09413187 0.25589971]
 [0.5331567  0.54625858 0.04505245 0.15143863]
 [0.57535932 0.59101919 0.31486483 0.19852598]
 [0.03896005 0.00282869 0.05588928 0.81589896]]
Multiplicación de dos matrices A(3x3)*B(3x2)
Matriz A(3x3):
[[2 3 4]
 [4 5 7]
 [3 2 1]]
Matriz B(3x2):
[[1 5]
 [4 3]
 [7 9]]
Matriz C=A*B(3x2)
[[42 55]
 [73 98]
 [18 30]]
Matriz inversa de A:
[[-1.8  1.   0.2]
 [ 3.4 -2.   0.4]
 [-1.4  1.  -0.4]]
Matriz identidad I=A@A^(-1)
A@B=dot(A,B)
[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-6.66133815e-16  1.00000000e+00  0.00000000e+00]
 [-2.22044605e-16 -2.22044605e-16  1.00000000e+00]]
Matriz ortogonal Q:
[[-0.37139068 -0.4835947  -0.79259392]
 [-0.74278135 -0.35743956  0.56613852]
 [-0.55708601  0.79898255 -0.22645541]]
Matriz triangular superior R:
[[-5.38516481 -5.94225082 -7.24211819]
 [ 0.         -1.64001682 -3.63747321]
 [ 0.          0.          0.56613852]]
Matriz A=Q@R:
[[2. 3. 4.]
 [4. 5. 7.]
 [3. 2. 1.]]
Determinante de A:  5.000000000000001
Se verifica det(A)=Prod(r[i,i]):
Determinante de A es: 5.000000000000005
```

4.2 Tablas de las principales funciones de arreglos en *Numpy*.

Tabla 1. Funciones unitarias.

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating-point, or complex values
`sqrt`	Compute the square root of each element (equivalent to `arr ** 0.5`)
`square`	Compute the square of each element (equivalent to `arr ** 2`)
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)
`ceil`	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
`floor`	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as a separate array
`isnan`	Return boolean array indicating whether each value is `NaN` (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-`NaN`) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise (equivalent to `~arr`).

Tabla 2. Funciones binarias.

Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum; `fmax` ignores `NaN`
`minimum, fmin`	Element-wise minimum; `fmin` ignores `NaN`
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument
`greater, greater_equal, less, less_equal, equal, not_equal`	Perform element-wise comparison, yielding boolean array (equivalent to infix operators `>, >=, <, <=, ==, !=`)
`logical_and, logical_or, logical_xor`	Compute element-wise truth value of logical operation (equivalent to infix operators `& |, ^`)

Tabla 3. Métodos estadísticos básicos.

Method	Description
`sum`	Sum of all the elements in the array or along an axis; zero-length arrays have sum 0
`mean`	Arithmetic mean; zero-length arrays have NaN mean
`std, var`	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n)
`min, max`	Minimum and maximum
`argmin, argmax`	Indices of minimum and maximum elements, respectively
`cumsum`	Cumulative sum of elements starting from 0
`cumprod`	Cumulative product of elements starting from 1

Tabla 4. Operaciones de conjuntos con arreglos.

Method	Description
`unique(x)`	Compute the sorted, unique elements in x
`intersect1d(x, y)`	Compute the sorted, common elements in x and y
`union1d(x, y)`	Compute the sorted union of elements
`in1d(x, y)`	Compute a boolean array indicating whether each element of x is contained in y
`setdiff1d(x, y)`	Set difference, elements in x that are not in y
`setxor1d(x, y)`	Set symmetric differences; elements that are in either of the arrays, but not both

Tabla 5. Funciones principales del álgebra lineal de Numpy.

Function	Description
`diag`	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
`dot`	Matrix multiplication
`trace`	Compute the sum of the diagonal elements
`det`	Compute the matrix determinant
`eig`	Compute the eigenvalues and eigenvectors of a square matrix
`inv`	Compute the inverse of a square matrix
`pinv`	Compute the Moore-Penrose pseudo-inverse of a matrix
`qr`	Compute the QR decomposition
`svd`	Compute the singular value decomposition (SVD)
`solve`	Solve the linear system Ax = b for x, where A is a square matrix
`lstsq`	Compute the least-squares solution to Ax = b

Tabla 6. Funciones de distribuciones de probabilidad y aleatorias.

Function	Description
`seed`	Seed the random number generator
`permutation`	Return a random permutation of a sequence, or return a permuted range
`shuffle`	Randomly permute a sequence in-place
`rand`	Draw samples from a uniform distribution
`randint`	Draw random integers from a given low-to-high range
`randn`	Draw samples from a normal distribution with mean 0 and standard deviation 1 (MATLAB-like interface)
`binomial`	Draw samples from a binomial distribution
`normal`	Draw samples from a normal (Gaussian) distribution
`beta`	Draw samples from a beta distribution
`chisquare`	Draw samples from a chi-square distribution
`gamma`	Draw samples from a gamma distribution
`uniform`	Draw samples from a uniform [0, 1) distribution

4.2 Biblioteca Pandas

Pandas es una biblioteca externa que se utiliza ampliamente para el análisis de datos y que se apoya en las bibliotecas *Numpy* y *Scypy*; opera, principalmente, con dos estructuras de datos muy usados: las *Series* y los *DataFrame.* También permite acceso de bases de datos, archivos externos de texto, archivos de *Excel*, etc.

4.2.1 Series

Una *Serie* es un arreglo unidimensional que contiene una secuencia de valores y un arreglo asociado de etiquetas de datos, llamados *índices*. Estos índices pueden ser numéricos o textos (tipo *string*).

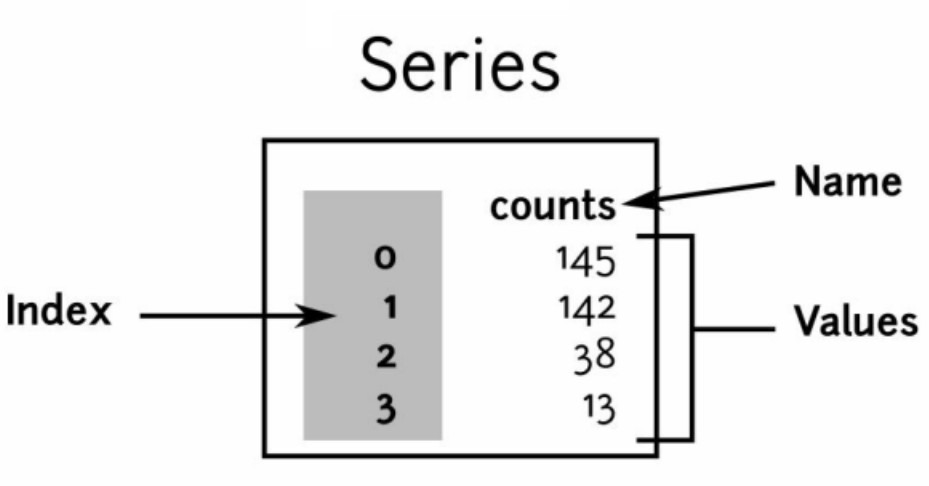


Figura 4.1 Partes de una serie

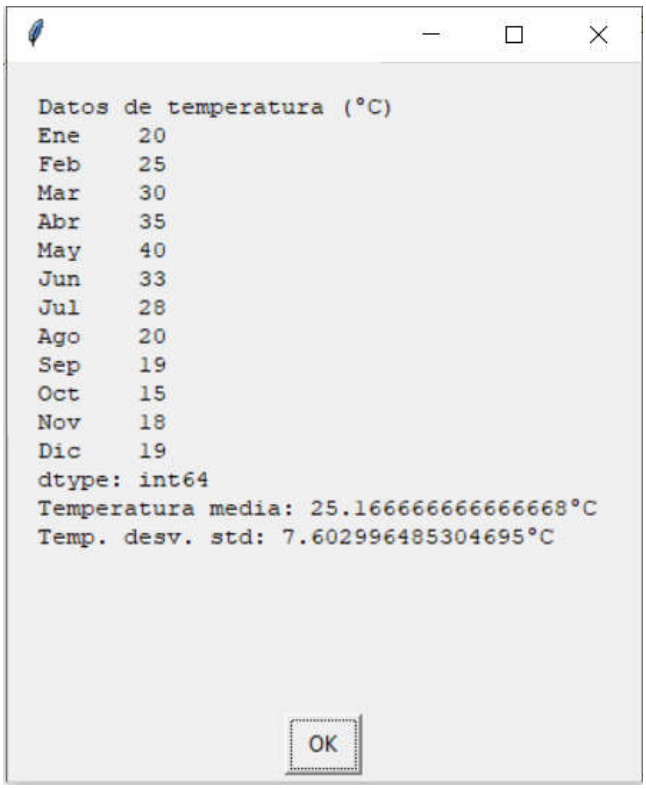
Ejemplo: Elaborar un programa para calcular la media y la desviación estándar de una serie de datos que representa a las temperaturas de un lugar registrados durante un año. Graficar su histograma.

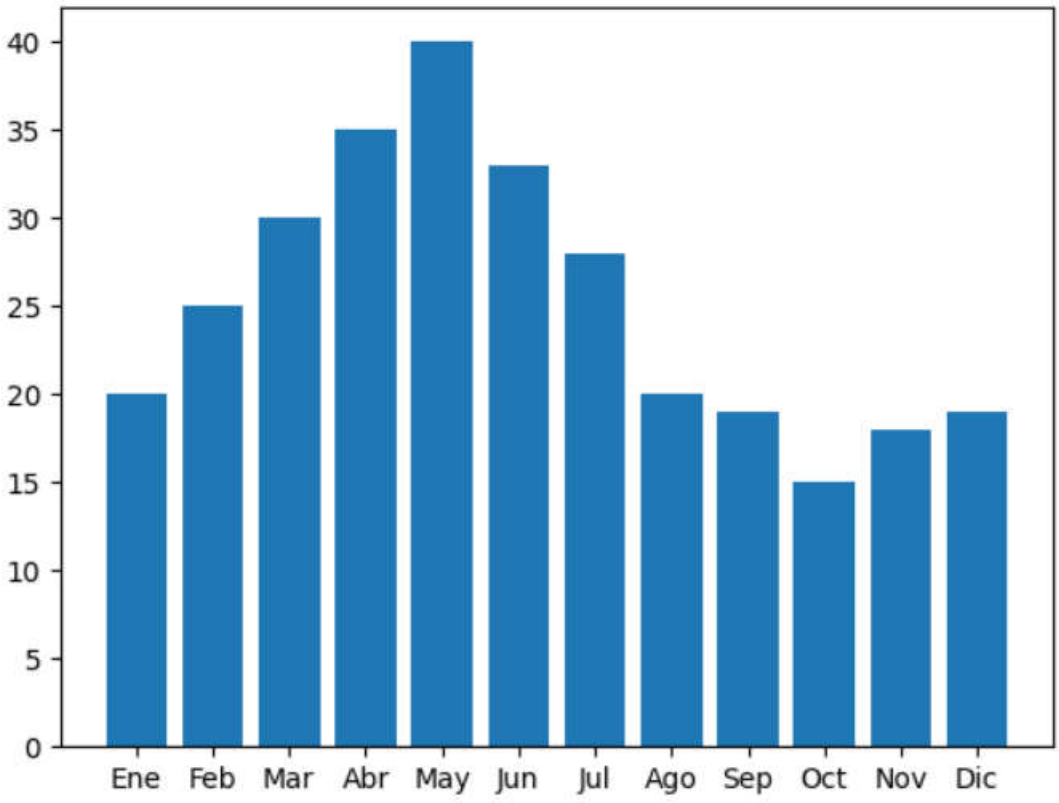
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as py
from tabulate import tabulate as tab
import easygui as eg

temperatura=pd.Series([20,25,30,35,40,33,28,20,19,15,18,19],
                index=["Ene","Feb","Mar",
                       "Abr","May","Jun",
                        "Jul","Ago","Sep",
                        "Oct","Nov","Dic"])

#tabla=tab([temperatura.index,temperatura.values])
#eg.msgbox(tabla)
tempmedia=np.mean(temperatura.values)
tempstd=np.std(temperatura.values)
eg.msgbox("Datos de temperatura (°C)\n"+str(temperatura)+
          "\nTemperatura media: "+str(tempmedia)+"°C\n"+
          "Temp. desv. std: "+str(tempstd)+"°C")

py.bar(temperatura.index,temperatura.values)
py.show()
```


Datos de temperatura (°C)
Ene 20
Feb 25
Mar 30
Abr 35
May 40
Jun 33
Jul 28
Ago 20
Sep 19
Oct 15
Nov 18
Dic 19
dtype: int64
Temperatura media: 25.166666666666668°C
Temp. desv. std: 7.602996485304695°C
OK


40
35
30
25
20
15
10
5
0
Ene
Feb
Mar
Abr
May
Jun
Jul
Ago
Sep
Oct
Nov
Dic

4.2.2 DataFrame

Un *DataFrame* representa una tabla rectangular de datos y contiene una colección ordenada de columnas, donde cada una puede ser de un tipo diferente de valor (numérico, texto, boleano, etc.) Los *DataFrame* tiene índices tanto en las filas como en las columnas. Los diccionarios de *Python* pueden servir para crear *DataFrames*.

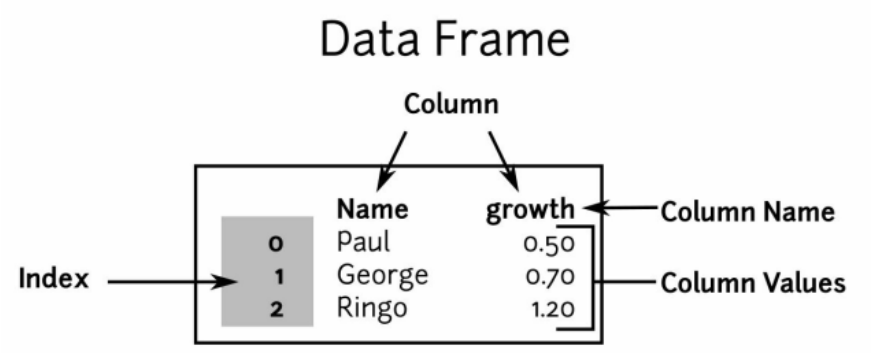


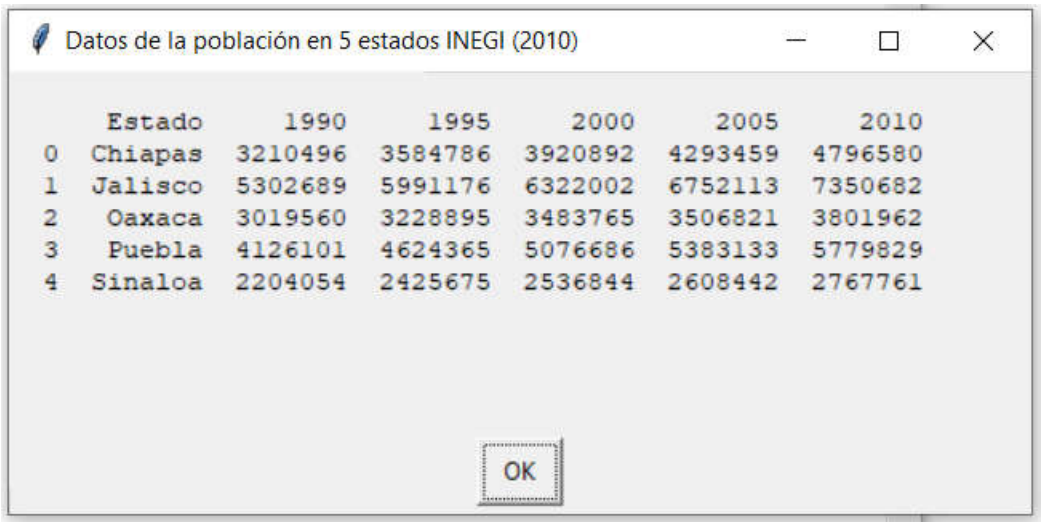
Figura 4.2 Elementos de un DataFrame.

Por ejemplo, los datos de la siguiente tabla, pueden ser cargados y mostrados en un *DataFrame* como se muestra en el programa siguiente:

Entidad federativa	Grupo quinquenal de edad	1990	1995	2000	2005	2010
		Total	Total	Total	Total	Total
Chiapas	Total	3 210 496	3 584 786	3 920 892	4 293 459	4 796 580
Jalisco	Total	5 302 689	5 991 176	6 322 002	6 752 113	7 350 682
Oaxaca	Total	3 019 560	3 228 895	3 438 765	3 506 821	3 801 962
Puebla	Total	4 126 101	4 624 365	5 076 686	5 383 133	5 779 829
Sinaloa	Total	2 204 054	2 425 675	2 536 844	2 608 442	2 767 761

```
import pandas as pd
import easygui as eg

poblacion={"Estado":["Chiapas","Jalisco","Oaxaca","Puebla","Sinaloa"],
           "1990":[3210496,5302689,3019560,4126101,2204054],
           "1995":[3584786,5991176,3228895,4624365,2425675],
           "2000":[3920892,6322002,3483765,5076686,2536844],
           "2005":[4293459,6752113,3506821,5383133,2608442],
           "2010":[4796580,7350682,3801962,5779829,2767761]}
TablaPob=pd.DataFrame(poblacion)
eg.msgbox(TablaPob,"Datos de la población en 5 estados INEGI (2010)")
```



A continuación, se presentan varias tablas que contienen instrucciones para las operaciones de los *DataFrame*, sin ahondar en el uso de ellas.

Tabla 7. Entradas posibles de datos para el constructor DataFrame.

Type	Notes
2D ndarray	A matrix of data, passing optional row and column labels
dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame; all sequences must be the same length
NumPy structured/record array	Treated as the "dict of arrays" case
dict of Series	Each value becomes a column; indexes from each Series are unioned together to form the result's row index if no explicit index is passed
dict of dicts	Each inner dict becomes a column; keys are unioned to form the row index as in the "dict of Series" case
List of dicts or Series	Each item becomes a row in the DataFrame; union of dict keys or Series indexes become the DataFrame's column labels
List of lists or tuples	Treated as the "2D ndarray" case
Another DataFrame	The DataFrame's indexes are used unless different ones are passed
NumPy MaskedArray	Like the "2D ndarray" case except masked values become NA/missing in the DataFrame result

Tabla 8. Algunos métodos Index y sus propiedades.

Method	Description
`append`	Concatenate with additional Index objects, producing a new Index
`difference`	Compute set difference as an Index
`intersection`	Compute set intersection
`union`	Compute set union
`isin`	Compute boolean array indicating whether each value is contained in the passed collection
`delete`	Compute new Index with element at index `i` deleted
`drop`	Compute new Index by deleting passed values
`insert`	Compute new Index by inserting element at index `i`
`is_monotonic`	Returns `True` if each element is greater than or equal to the previous element
`is_unique`	Returns `True` if the Index has no duplicate values
`unique`	Compute the array of unique values in the Index

Tabla 9. Argumentos de la función de reindexado.

Argument	Description
`index`	New sequence to use as index. Can be Index instance or any other sequence-like Python data structure. An Index will be used exactly as is without any copying.
`method`	Interpolation (fill) method; `'ffill'` fills forward, while `'bfill'` fills backward.
`fill_value`	Substitute value to use when introducing missing data by reindexing.
`limit`	When forward- or backfilling, maximum size gap (in number of elements) to fill.
`tolerance`	When forward- or backfilling, maximum size gap (in absolute numeric distance) to fill for inexact matches.
`level`	Match simple Index on level of MultiIndex; otherwise select subset of.
`copy`	If `True`, always copy underlying data even if new index is equivalent to old index; if `False`, do not copy the data when the indexes are equivalent.

Tabla 10. Opciones de indexación con DataFrame.

Type	Notes
`df[val]`	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
`df.loc[val]`	Selects single row or subset of rows from the DataFrame by label
`df.loc[:, val]`	Selects single column or subset of columns by label
`df.loc[val1, val2]`	Select both rows and columns by label
`df.iloc[where]`	Selects single row or subset of rows from the DataFrame by integer position

Tabla 11. Métodos aritméticos flexibles.

Method	Description
`add, radd`	Methods for addition (+)
`sub, rsub`	Methods for subtraction (-)
`div, rdiv`	Methods for division (/)
`floordiv, rfloordiv`	Methods for floor division (//)
`mul, rmul`	Methods for multiplication (*)
`pow, rpow`	Methods for exponentiation (**)

Tabla 12. Métodos de desempate (tie-breaking) con rangos.

Method	Description
`'average'`	Default: assign the average rank to each entry in the equal group
`'min'`	Use the minimum rank for the whole group
`'max'`	Use the maximum rank for the whole group
`'first'`	Assign ranks in the order the values appear in the data
`'dense'`	Like `method='min'`, but ranks always increase by 1 in between groups rather than the number of equal elements in a group

Tabla 13. Opciones para los métodos de reducción.

Method	Description
`axis`	Axis to reduce over; 0 for DataFrame's rows and 1 for columns
`skipna`	Exclude missing values; `True` by default
`level`	Reduce grouped by level if the axis is hierarchically indexed (MultiIndex)

Tabla 14. Estadística descriptiva y sumatoria.

Method	Description
`count`	Number of non-NA values
`describe`	Compute set of summary statistics for Series or each DataFrame column
`min, max`	Compute minimum and maximum values
`argmin, argmax`	Compute index locations (integers) at which minimum or maximum value obtained, respectively
`idxmin, idxmax`	Compute index labels at which minimum or maximum value obtained, respectively
`quantile`	Compute sample quantile ranging from 0 to 1
`sum`	Sum of values
`mean`	Mean of values
`median`	Arithmetic median (50% quantile) of values
`mad`	Mean absolute deviation from mean value
`prod`	Product of all values
`var`	Sample variance of values
`std`	Sample standard deviation of values
`skew`	Sample skewness (third moment) of values
`kurt`	Sample kurtosis (fourth moment) of values
`cumsum`	Cumulative sum of values
`cummin, cummax`	Cumulative minimum or maximum of values, respectively
`cumprod`	Cumulative product of values
`diff`	Compute first arithmetic difference (useful for time series)
`pct_change`	Compute percent changes

Tabla 15. Formatos de archivos externos para lectura/escritura.

Function	Description
read_csv	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
read_table	Load delimited data from a file, URL, or file-like object; use tab ('\t') as default delimiter
read_fwf	Read data in fixed-width column format (i.e., no delimiters)
read_clipboard	Version of read_table that reads data from the clipboard; useful for converting tables from web pages
read_excel	Read tabular data from an Excel XLS or XLSX file
read_hdf	Read HDF5 files written by pandas
read_html	Read all tables found in the given HTML document
read_json	Read data from a JSON (JavaScript Object Notation) string representation
read_msgpack	Read pandas data encoded using the MessagePack binary format
read_pickle	Read an arbitrary object stored in Python pickle format
read_sas	Read a SAS dataset stored in one of the SAS system's custom storage formats
read_sql	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
read_stata	Read a dataset from Stata file format
read_feather	Read the Feather binary file format

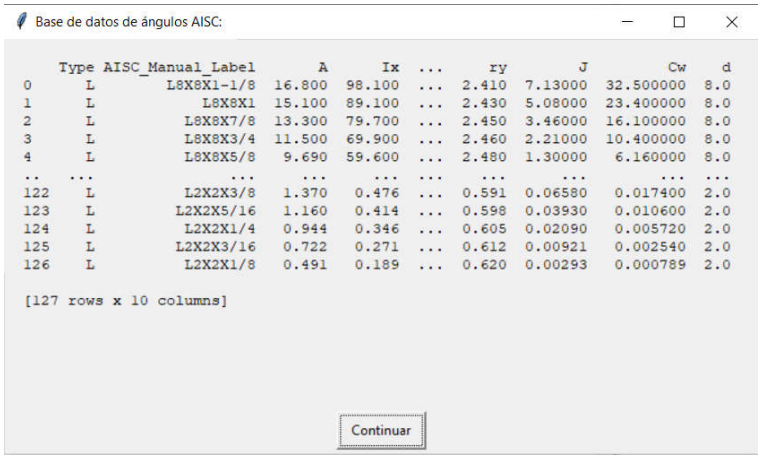
Pandas soporta la lectura tabular de datos de archivos de *Excel* usando la clase *ExcelFile* y la función *pandas.read_excel()*. Internamente estas herramientas usan los paquetes externos *xlrd* y *openpyxl* para leer archivos *XLS* y *XLSX*, respectivamente. Estos paquetes o bibliotecas deben instalarse previamente con *pip*.

En la figura 10 se muestra la instalación de la biblioteca externa o paquete xlrd.

Figura 10. Instalación de xlrd

El siguiente programa lee un archivo *Excel* ubicado en el mismo directorio o carpeta donde se encuentra el programa fuente que contiene los datos geométricos del perfil L (ángulo) de la base de datos *AISC*, se pasa a una variable *ArchivoXLS* y se muestra en una ventana de *EasyGUI*.

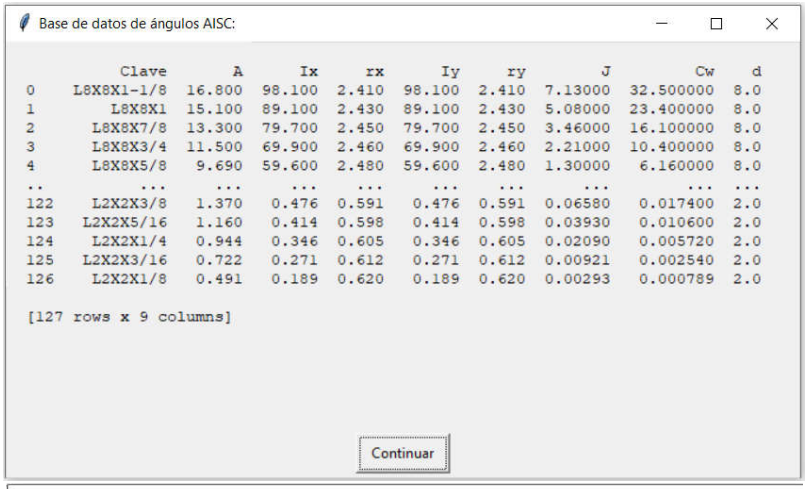
```
import pandas as pd
import easygui as eg
xlsx=pd.ExcelFile("AISC_ANGULOS_2003.XLS")
ArchivoXLS=pd.read_excel(xlsx,"Hoja1")
eg.msgbox(ArchivoXLS,"Base de datos de ángulos AISC:","Lectura")
```

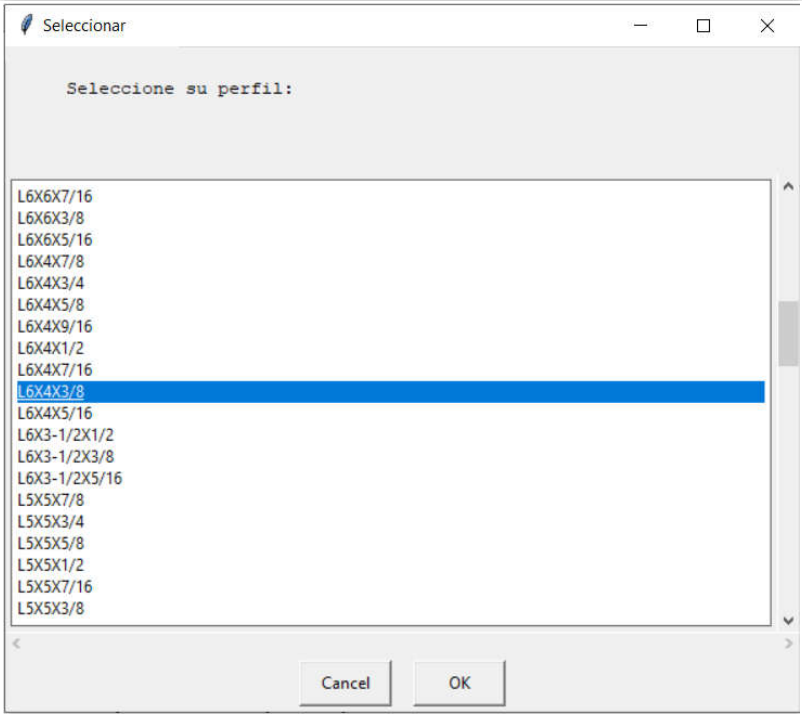


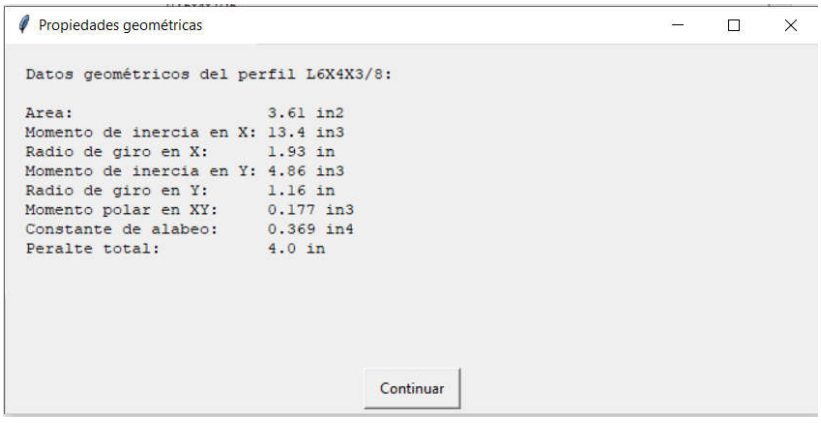
Usando este archivo, haremos una aplicación que permita mostrar las propiedades geométricas de un perfil L seleccionado de una lista de perfiles.

```
import pandas as pd
import easygui as eg
def Inicio():
    xlsx=pd.ExcelFile("AISCANGULOS2003.XLS")
    ArchivoXLS=pd.read_excel(xlsx,"Hoja1")
    clave=ArchivoXLS["Clave"]
    PerfiL=ArchivoXLS.set_index("Clave")
    eg.msgbox(ArchivoXLS,"Base de datos de ángulos AISC:","Continuar")
    return clave,PerfiL
def Escoger(clave):
    perfil=eg.choicebox("Seleccione su perfil:","Seleccionar",clave)
    return perfil
def Propiedades(PerfiL,perfil):
    Area=PerfiL["A"][perfil]
    InX=PerfiL["Ix"][perfil]
    raX=PerfiL["rx"][perfil]
    InY=PerfiL["Iy"][perfil]
    raY=PerfiL["ry"][perfil]
    JXY=PerfiL["J"][perfil]
    CWp=PerfiL["Cw"][perfil]
    dLL=PerfiL["d"][perfil]
    Areatxt=str(round(Area,3))+" in2\n"
    InXtxt=str(round(InX,3))+" in3\n"
    raXtxt=str(round(raX,3))+" in\n"
    InYtxt=str(round(InY,3))+" in3\n"
    raYtxt=str(round(raY,3))+" in\n"
    JXYtxt=str(round(JXY,3))+" in3\n"
    Cwptxt=str(round(CWp,3))+" in4\n"
    dLLtxt=str(round(dLL,3))+" in"
    txt=""
    txt=txt+"Datos geométricos del perfil "+perfil+":\n\n"
    txt=txt+"Area:                    "+Areatxt
    txt=txt+"Momento de inercia en X: "+InXtxt
    txt=txt+"Radio de giro en X:      "+raXtxt
    txt=txt+"Momento de inercia en Y: "+InYtxt
    txt=txt+"Radio de giro en Y:      "+raYtxt
    txt=txt+"Momento polar en XY:     "+JXYtxt
    txt=txt+"Constante de alabeo:     "+Cwptxt
    txt=txt+"Peralte total:           "+dLLtxt
    eg.msgbox(txt,"Propiedades geométricas","Continuar")

clave,PerfiL=Inicio()
while True:
    perfil=Escoger(clave)
    Propiedades(PerfiL,perfil)
    otro=eg.ynbox("Otro (Yes/No): ")
    if otro==False:
        break
```


Base de datos de ángulos AISC:
Clave A Ix rx Iy ry J Cw d
0 L8X8X1-1/8 16.800 98.100 2.410 98.100 2.410 7.13000 32.500000 8.0
1 L8X8X1 15.100 89.100 2.430 89.100 2.430 5.08000 23.400000 8.0
2 L8X8X7/8 13.300 79.700 2.450 79.700 2.450 3.46000 16.100000 8.0
3 L8X8X3/4 11.500 69.900 2.460 69.900 2.460 2.21000 10.400000 8.0
4 L8X8X5/8 9.690 59.600 2.480 59.600 2.480 1.30000 6.160000 8.0
122 L2X2X3/8 1.370 0.476 0.591 0.476 0.591 0.06580 0.017400 2.0
123 L2X2X5/16 1.160 0.414 0.598 0.414 0.598 0.03930 0.010600 2.0
124 L2X2X1/4 0.944 0.346 0.605 0.346 0.605 0.02090 0.005720 2.0
125 L2X2X3/16 0.722 0.271 0.612 0.271 0.612 0.00921 0.002540 2.0
126 L2X2X1/8 0.491 0.189 0.620 0.189 0.620 0.00293 0.000789 2.0
[127 rows x 9 columns]
Continuar


Seleccionar
Seleccione su perfil:
L6X6X7/16
L6X6X3/8
L6X6X5/16
L6X4X7/8
L6X4X3/4
L6X4X5/8
L6X4X9/16
L6X4X1/2
L6X4X7/16
L6X4X3/8
L6X4X5/16
L6X3-1/2X1/2
L6X3-1/2X3/8
L6X3-1/2X5/16
L5X5X7/8
L5X5X3/4
L5X5X5/8
L5X5X1/2
L5X5X7/16
L5X5X3/8
Cancel
OK


Propiedades geométricas
Datos geométricos del perfil L6X4X3/8:
Area: 3.61 in2
Momento de inercia en X: 13.4 in3
Radio de giro en X: 1.93 in
Momento de inercia en Y: 4.86 in3
Radio de giro en Y: 1.16 in
Momento polar en XY: 0.177 in3
Constante de alabeo: 0.369 in4
Peralte total: 4.0 in
Continuar