

Guía Ciencia de Datos

Joaquín Barrutia

SQLAlchemy

SQLAlchemy es una librería utilizada para crear bases de datos y manipularlas en MySQL y otros sistemas de gestión de bases de datos desde Python, esto es posible mediante el Mapeo Objeto-Relacional (ORM). Un ORM es un modelo de programación que permite mapear las estructuras de una base de datos relacional como SQL Server, Oracle, MySQL sobre una estructura lógica de entidades con el objeto de simplificar el desarrollo de nuestras aplicaciones. Lo más común es que una tabla se corresponda con una clase, las filas de una tabla con objetos, las columnas con atributos de clase y las *Foreign Keys* con relaciones entre clases.

Uso de SQLAlchemy

Para hacer uso de cualquier aplicación de SQLAlchemy lo primero que necesitamos hacer es crear un objeto llamado `engine`. Para crear este objeto tenemos que utilizar `create_engine()`.

```
from sqlalchemy import create_engine
#1 especificamos el dialecto, 2 usuario y contraseña, 3 localhos o
ip, 4 base
engine = create_engine('mysql+pymysql://usuario:contraseña@localhos
t/base_ejemplo')
```

Una vez que hemos creado un `engine` debemos crear una sesión (`session`) utilizando `sessionmaker()` y el `engine` asociado que creamos en el paso anterior.

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()
```

Posteriormente, tenemos que crear el modelo (o clase) equivalente a la tabla que queremos replicar. Creamos una clase llamada `Base` utilizando `declarative_base()`, de esta clase heredarán todos los modelos.

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

De esta manera podemos unir todos los pasos anteriores en un solo bloque de código:

```

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine('mysql+pymysql://usuario:contraseña@localhost/base_ejemplo')

Session = sessionmaker(bind=engine)
session = Session()
Base = declarative_base()

```

El paso a seguir es crear un modelo `Tabla1`. Este será un ejemplo de la implementación de un molde para una tabla. Tenemos que importar las propiedades que necesitemos dependiendo de nuestra tabla.

```

#importar propiedades necesarias
from sqlalchemy import Column, Integer, String, Float, CHAR, Date, Boolean, CheckConstraint

class Tabla1(Base): # hereda de Base
    __tablename__ = 'tabla1' #asignamos nombre
    __table_args__ = (
        CheckConstraint('atributo2 > 0', name = 'check1'), #restricción tipo check
    )

    id = Column(Integer, primary_key=True) #Llave primaria
    atributo1 = Column(String(25), nullable=False)
    atributo2 = Column(Float)
    atributo3 = Column(Float)
    atributo4 = Column(String)
    atributo5 = Column(Date)

    def __init__(self, atributo1, atributo2, atributo3, atributo4, atributo5):
        self.atributo1 = atributo1
        self.atributo2 = atributo2
        self.atributo3 = atributo3
        self.atributo4 = atributo4
        self.atributo5 = atributo5

```

En este ejemplo la clase `Tabla1` representa un ejemplo de una tabla cualquiera. Cada una de las columnas de la tabla está representada por atributos de tipo `Column` (e.g., `atributo1`). Para cada una de estas columnas definimos distintos tipos de datos (`Integer`, `String`, `Float`, etc.). Adicionalmente definimos una restricción de tipo `CHECK`, donde debe cumplirse que `atributo2 > 0` y la nombramos `check1`. Podemos agregar más restricciones dentro de `__table_args__`. Por último también se puede notar que definimos a `id` como nuestra llave primaria atómica.

Si quisieramos crear otra tabla y asignarle una llave foránea que haga referencia a nuestra Tabla1 tenemos que crear una nueva clase de una manera similar a como lo hicimos anteriormente:

```
class tabla2(Base):
    __tablename__ = 'tabla2'

    id_1 = Column(Integer, primary_key=True) #Llave primaria compuesta
    id_2 = Column(Integer, primary_key=True) #Llave primaria compuesta
    usuario = Column(Integer, ForeignKey('tabla1.id'))
    #otra forma de agregar restricciones
    atributo2 = Column(Float, CheckConstraint('atributo2 < 100', name = 'check2'))
    atributo3 = Column(String)

    def __init__(self, atributo1, atributo2, atributo3, atributo4, atributo5):
        self.atributo1 = atributo1
        self.atributo2 = atributo2
        self.atributo3 = atributo3
        self.atributo4 = atributo4
        self.atributo5 = atributo2
```

En este ejemplo estamos asignando a `usuario` como una llave foránea que hace referencia al `id` de la `Tabla1`. Adicionalmente, definimos `id_1` y `id_2` como nuestra llave primaria compuesta.

Consultas

Para realizar cualquier consulta a nuestras bases de datos son necesarios dos elementos: un `engine` para conectarnos como vimos anteriormente y la librería `pandas`.

```
from sqlalchemy import create_engine
import pandas as pd
```

Creamos nuestro `engine`.

```
from sqlalchemy import create_engine
#1 especificamos el dialecto, 2 usuario y contraseña, 3 localhos o ip, 4 base
engine = create_engine('mysql+pymysql://usuario:contraseña@localhost/base_ejemplo')
```

Leemos nuestra tabla creada en MySQL.

```
tabla1 = pd.read_sql("Select * from tabla1",engine) # el * es para
```

NumPY

NumPy es una librería cuyo principal objetivo son los arreglos homogéneos multidimensionales. La clase de arreglos en Numpy se conocen como `ndarrays` o más comúnmente, `arrays`. Las dimensiones en numpy se llaman `axes`; el número de ejes se llama `rank`.

En otras palabras, si consideramos el siguiente vector:

$$\mathbf{x} \in \mathbb{R}^3$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Estaríamos diciendo que `x` tiene 3 elementos, cada uno de ellos números reales. Y lo podríamos representar en Python con el siguiente `numpy array`.

```
import numpy as np

x = np.array([2.5, 3.2, 1.0])
```

A continuación algunos ejemplos de `numpy arrays` que podemos implementar.

```
In [2]: #Array del 0 al 9
import numpy as np

x_1 = np.arange(10)
x_1
```

```
Out[2]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [3]: #Array de unos

x_2 = np.ones(5)
x_2
```

```
Out[3]: array([1., 1., 1., 1., 1.])
```

```
In [5]: #Array de ceros

x_3 = np.zeros(4)
x_3
```

```
Out[5]: array([0., 0., 0., 0.])
```

```
In [6]: # Vector fila
v_fila = np.array([1, 2, 3])

# Vector columna
v_col = np.array([ [1], [2], [3]])

print(v_fila)
print(v_col)
```

```
[1 2 3]
[[1]
 [2]
 [3]]
```

También es posible representar matrices utilizando un la librería de **NumPy**.

Es decir la matriz:

$$\mathbf{X} \in \mathbb{R}^{3 \times 3}$$

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$$

La podemos representar en Python de la siguiente manera:

```
X = np.array([
    [1.2, 0.5, 0.0],
    [2.4, 1.0, 7.8],
    [2.1, 3.2, 5.5]
])
```

De la misma forma que lo hicimos con los vectores a continuación se muestran algunos ejemplos.

```
In [8]: # podemos crear una matriz de 3X3 con la función de reshape la cual toma co
x = np.arange(9).reshape((3,3))
x
```

```
Out[8]: array([[0, 1, 2],
               [3, 4, 5],
               [6, 7, 8]])
```

```
In [9]: #matrices de ceros

np.zeros((5,5))
```

```
Out[9]: array([[0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0.]])
```

```
In [10]: #matrices de unos  
np.ones((4,4))
```

```
Out[10]: array([[1., 1., 1., 1.],  
                [1., 1., 1., 1.],  
                [1., 1., 1., 1.],  
                [1., 1., 1., 1.]])
```

```
In [11]: #Matriz identidad  
np.identity(5)
```

```
Out[11]: array([[1., 0., 0., 0., 0.],  
                [0., 1., 0., 0., 0.],  
                [0., 0., 1., 0., 0.],  
                [0., 0., 0., 1., 0.],  
                [0., 0., 0., 0., 1.]])
```

```
In [12]: # podemos conocer la forma de nuestra matriz  
x.shape
```

```
Out[12]: (3, 3)
```

```
In [13]: #Podemos implemntar matrices aleatorias  
# en este caso es una matriz con números enteros entre 0 y 10 de forma 3X3  
np.random.randint(0,10,size=(3,3))
```

```
Out[13]: array([[6, 7, 6],  
                [4, 1, 5],  
                [8, 6, 7]])
```

NumPy también nos sirve para simular distintas distribuciones de probabilidad, es necesario establecer los parámetros de la función y el número de elementos que se van a generar. A continuación algunos ejemplos.

```
In [15]: bernoulli = np.random.binomial(1, 0.5,10) # simular un volado 10 veces
print(bernoulli)
print()
binomial = np.random.binomial(10, 0.5,100) #simular 10 volados 100 veces
print(binomial)
print()
poisson = np.random.poisson(2, 20) #poisson
print(poisson)
print()
unifrome = np.random.uniform(1,0,20) #distribución uniforme
print(unifrome)
print()
exponencial = np.random.exponential(1, 20) #distribución exponencial
print(exponencial)
print()
normal = np.random.normal(5, 3, 20) #distribución normal
print(normal)
print()
```

```
[1 0 1 1 0 0 0 1 0 1]
```

```
[2 6 5 1 6 8 7 6 5 6 4 2 4 5 6 3 4 5 8 4 7 3 5 6 5 5 6 8 6 5 6 5 4 5 3 6
4
4 6 2 4 8 8 5 4 5 6 5 6 5 4 5 7 7 4 7 6 6 6 5 4 4 4 6 3 4 6 6 6 4 8 3 5
3
1 8 4 5 7 4 5 7 8 5 3 4 5 7 5 5 7 4 7 3 4 6 6 4 5 2]
```

```
[1 1 3 1 1 1 3 3 1 1 1 2 1 1 1 2 4 2 4 2]
```

```
[0.39965296 0.87775195 0.32796992 0.32869182 0.21359628 0.14659907
0.97272683 0.21971239 0.03197816 0.97085689 0.47403274 0.933887
0.66154171 0.17512668 0.93020573 0.95465767 0.42934021 0.45641548
0.34257887 0.7638324 ]
```

```
[0.30787346 3.58905317 0.57994739 1.05491987 1.17643095 0.22342707
1.2666127 1.03879733 0.20884338 0.65371737 1.10945515 1.07257296
2.38985174 0.0789866 0.75349639 1.02087753 1.19920527 0.4958011
0.90090131 0.09435468]
```

```
[ 7.75375564  4.46990638  3.8863438  12.00534838  8.90407527 -0.30366759
 7.78433908  8.00685045  9.0219523  9.64926548  4.60141939  3.98494313
 2.49937617  6.66103374  4.78367079  6.1818665  5.81677281  8.24102667
 3.30714408  7.8038165 ]
```

Operaciones entre arrays

Una de las características principales de los arrays es que cumplen las propiedades de las operaciones entre matrices como las conocemos. Esto permite realizar operaciones algebraicas entrada por entrada de manera más eficiente, es decir sin necesidad de realizar loops.

```
In [17]: #aquí suma 1 a cada entrada
np.arange(9).reshape((3,3)) + 1
```

```
Out[17]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

```
In [18]: #aquí multiplicamos por un escalar cada entrada
np.arange(9).reshape((3,3)) * 10
```

```
Out[18]: array([[ 0, 10, 20],
               [30, 40, 50],
               [60, 70, 80]])
```

Producto Punto

$$x \cdot y = \sum x_i y_i$$

```
In [19]: x = np.array([1,2,3])
y = np.array([4,5,6])

np.dot(x,y)
```

```
Out[19]: 32
```

```
In [23]: # Transponer matriz

print(np.arange(6).reshape((2,3)))
print()
print(np.arange(6).reshape((2,3)).T)
```

```
[[0 1 2]
 [3 4 5]]
```

```
[[0 3]
 [1 4]
 [2 5]]
```

```
In [32]: # Suma de matrices
a = np.arange(16).reshape((4,4))
b = np.ones((4,4))
a + b
```

```
Out[32]: array([[ 1.,  2.,  3.,  4.],
               [ 5.,  6.,  7.,  8.],
               [ 9., 10., 11., 12.],
               [13., 14., 15., 16.]])
```



```
In [35]: # multiplicación de matrices
a = np.arange(16).reshape((4,4))
b = np.ones((4,2))
a @ b
```

```
Out[35]: array([[ 6.,  6.],
                [22., 22.],
                [38., 38.],
                [54., 54.]])
```

Además de estos ejemplos existen muchas funciones adicionales en la librería de **NumPy** y muchas utilidades para la ciencia de datos. Podemos por ejemplo resolver sistemas de ecuaciones, encontrar determinantes de matrices, su inversa, etc.

SciPy

Es una librería de Python compuesta principalmente de herramientas y algoritmos matemáticos. **SciPy** contiene módulos para optimización, álgebra lineal, integración, interpolación, entre otros. Esta librería está basada en los objetos de *NumPy*.

Entre las herramientas más útiles de esta librería se encuentra la integración.

Integración

SciPy cuenta con una función que nos ayuda a aproximar integrales de una manera sencilla. A continuación se describe como aproximar una integral del estilo:

$$\int_a^b f(x) dx$$

Lo primero que debemos hacer es importar la librería.

```
from scipy import integrate
```

Posteriormente necesitamos definir la función que queremos integrar. Por ejemplo $f(x) = x^2$ lo definiríamos así:

```
def f(x):
    return (x**2)
```

Por último utilizamos la aproximación de **SciPy** de la siguiente manera:

```
integrate.quad(f, a, b)
```

Donde f es la función previamente definida, a es el límite inferior y b el límite superior sobre los que se hará la aproximación a la integral.

Podemos utilizar una función `lambda` y realizar todo lo anterior en una sola línea de código.

```
integrate.quad(lambda x: x**2, a, b)
```

```
In [39]: from scipy import integrate
integrate.quad(lambda x: x**2, 0, 10)
```

```
Out[39]: (333.3333333333326, 3.700743415417188e-12)
```

También es posible aproximar una doble integral de la siguiente manera:

```
#definimos funciones
f = lambda y, x: x*y**2
#aplicamos dblquad
integrate.dblquad(f, a, b, lambda x: 0, lambda x: 1)
```

También podríamos poner límites variables y más casos específicos.

Interpolación

Algunas veces nos encontraremos con una serie de datos para ciertas observaciones, pero nos harán falta algunas observaciones específicas. Si quisiéramos conocer el valor de estos datos necesitaríamos de alguna herramienta para inferirlos, para ello podemos utilizar la interpolación, es decir construir una curva que pase por todos los puntos en nuestra muestra para así con esta curva poder predecir los valores que nos hacen falta.

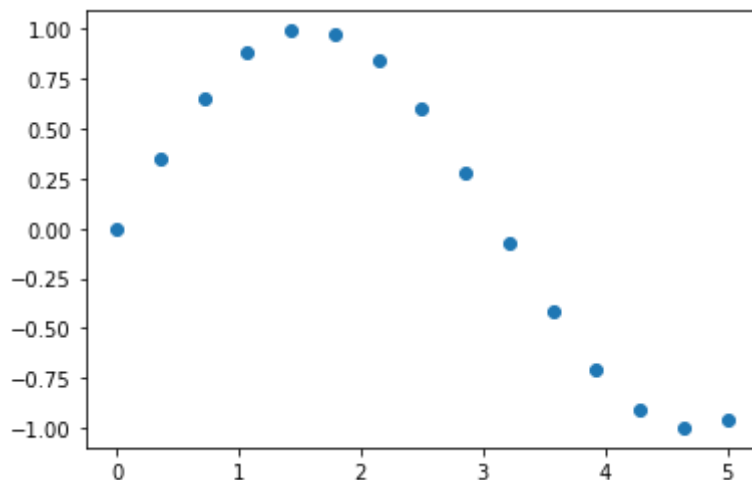
Es útil ver esto graficamente para entender el concepto.

Lo primero que haremos es definir algunos datos y los graficaremos en un scatter plot.

```
In [64]: import matplotlib.pyplot as plt

x = np.linspace(0, 5, 15)
y = np.sin(x)

plt.plot(x, y, 'o ')
plt.show()
```

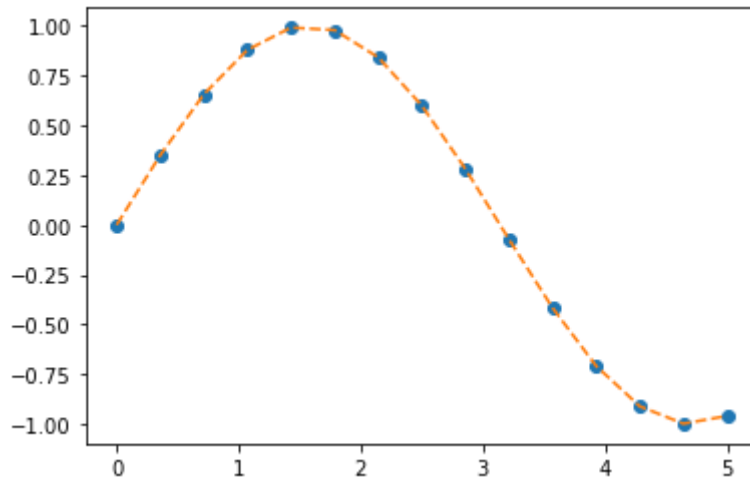


Ahora construiremos una curva que pase por todos estos puntos, con el objetivo de predecir valores con los que no contamos pero que se encuentran entre estos puntos. Utilizaremos la clase `interp1d` de `scipy.interpolate`.

```
In [65]: from scipy.interpolate import interp1d

f = interp1d(x,y)

plt.plot(x, y, 'o ')
plt.plot(x, f(x), '--')
plt.show()
```



Ahora ya teniendo nuestra curva podríamos predecir valores que se encuentran entre nuestros puntos.

```
In [66]: #Estos son nuestros datos originales
print(x)
print()
print(y)
```

```
[0.          0.35714286  0.71428571  1.07142857  1.42857143  1.78571429
 2.14285714  2.5         2.85714286  3.21428571  3.57142857  3.92857143
 4.28571429  4.64285714  5.          ]
```

```
[ 0.          0.34959881  0.6550779   0.8778855   0.98990308  0.97699389
 0.84078711  0.59847214  0.2806294  -0.07262906 -0.41672165 -0.70822356
-0.91034694 -0.99758364 -0.95892427]
```

si quisieramos predecir por ejemplo cual los valores de algunos datos con los que no contabamos originalmente haríamos lo siguiente:

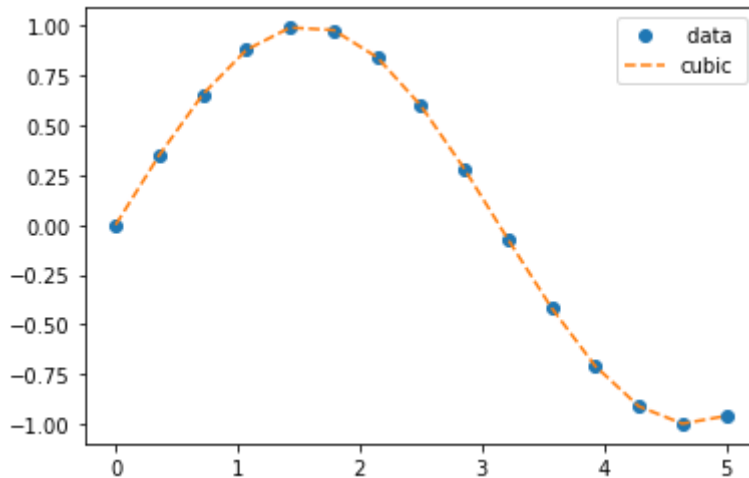
```
In [67]: f([1,2,3,4])
```

```
Out[67]: array([ 0.83332398,  0.89526982,  0.13932602, -0.74864824])
```

Es posible dentro de los argumentos de `interp1d` realizar interpolaciones polinómicas de distintos grados.

```
In [68]: f_2 = interp1d(x,y,kind="cubic")

plt.plot(x, y, 'o')
plt.plot(x, f_2(x), '--')
plt.legend(['data', 'cubic'])
plt.show()
```



Y podemos predecir los mismos valores con esta nueva curva.

```
In [70]: f_2([1,2,3,4])
```

```
Out[70]: array([ 0.84146335,  0.90925946,  0.14111755, -0.75679981])
```

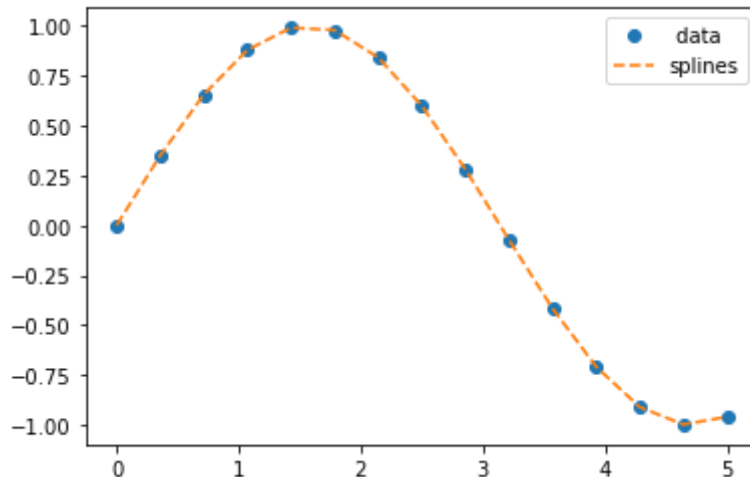
Splines

Un prolema al que nos podremos enfrentar es que cuando se quiere interpolar con un polinomio una función con n puntos, cuando n aumenta, también aumenta la oscilación entre puntos, por lo que no es una interpolación representativa, a este fenómeno se le conoce como el "fenómeno de Runge". Una posible solución es el uso de interpolación por splines, los splines son curvas polinómicas definidas a trozo.

```
In [73]: from scipy.interpolate import InterpolatedUnivariateSpline

fsp = InterpolatedUnivariateSpline(x, y)

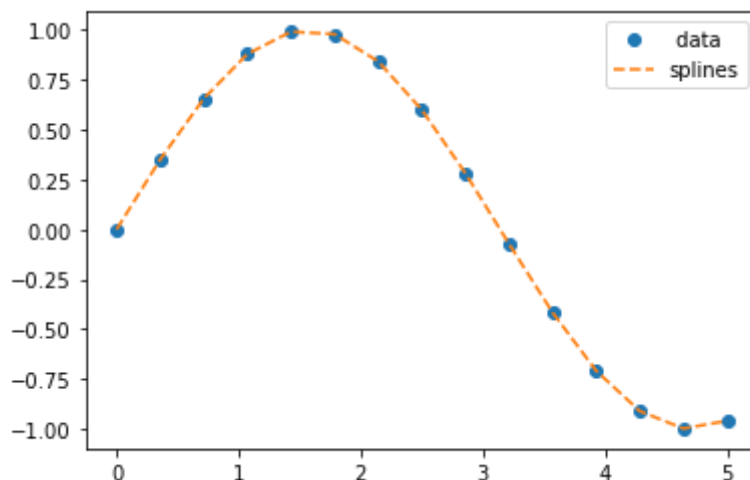
plt.plot(x, y, 'o')
plt.plot(x, fsp(x), '--')
plt.legend(['data', 'splines'])
plt.show()
```



Es posible dar como argumento el grado de los splines.

```
In [76]: fsp_2 = InterpolatedUnivariateSpline(x, y, k=4)

plt.plot(x, y, 'o')
plt.plot(x, fsp_2(x), '--')
plt.legend(['data', 'splines'])
plt.show()
```



Como se puede ver el ajuste es bueno independientemente de si utilizamos splines o interpolación polinómica, pero habrán casos más complejos donde se deberá elegir aquel modelo que ajuste de mejor manera.

Otra función que nos permite realizar **SciPy** es la obtención de las raíces de un polinomio.

La forma de realizar esto es la siguiente:

Lo primero que debemos hacer es generar un polinomio. Por ejemplo podemos crear el siguiente polinomio:

$$P(x) = x^3 + x^2 + x + c$$

```

In [79]: import scipy as sp

polinomio = [1,2,-5,-6]# polinomio =  $x^3 - 2x^2 - 5x + 6$ 

x = sp.arange(-5,5,.05) #areglo para valuar polinomio

y = np.polyval(polinomio,x) #evaluamos el polinomio

raices = sp.roots(polinomio) #calculamos raíces

s = sp.polyval(polinomio,raices)#evaluamos el polinomio en las raices

print (f"Raices: {np.round(raices[0],2)}, {np.round(raices[1],2)}, {np.roun

plt.figure

plt.plot(x,y,'-', label = 'y(x)')

plt.plot(raices,s,'o', label = 'Raices')
plt.xlabel('x')
plt.ylabel('y')
plt.title(f'Raices de un polinomio grado {[len(polinomio)-1]}')
# Leyenda
plt.legend()
# Mostramos la figura en pantalla
plt.show()

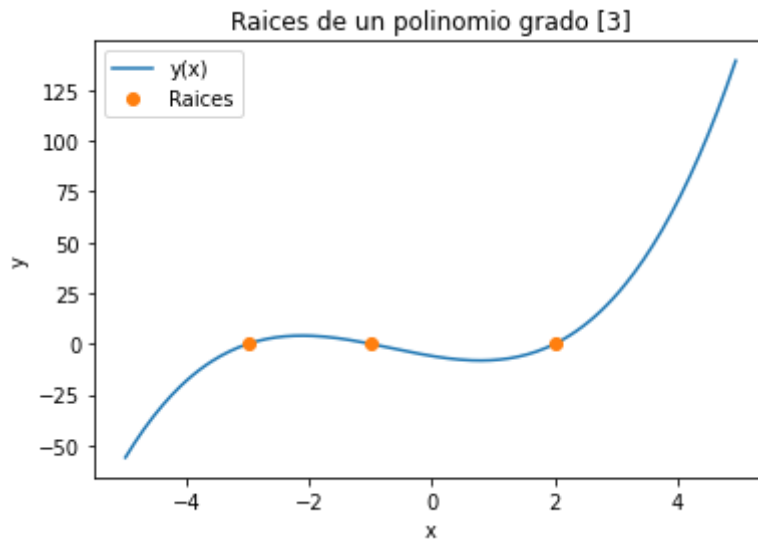
```

Raices: 2.0, -3.0, -1.0

```

/var/folders/l_/pc12z1g16z359gcfwljgy3d00000gn/T/ipykernel_54900/38915955
43.py:5: DeprecationWarning: scipy.arange is deprecated and will be remov
ed in SciPy 2.0.0, use numpy.arange instead
    x = sp.arange(-5,5,.05) #areglo para valuar polinomio
/var/folders/l_/pc12z1g16z359gcfwljgy3d00000gn/T/ipykernel_54900/38915955
43.py:9: DeprecationWarning: scipy.roots is deprecated and will be remove
d in SciPy 2.0.0, use numpy.roots instead
    raices = sp.roots(polinomio) #calculamos raíces
/var/folders/l_/pc12z1g16z359gcfwljgy3d00000gn/T/ipykernel_54900/38915955
43.py:11: DeprecationWarning: scipy.polyval is deprecated and will be rem
oved in SciPy 2.0.0, use numpy.polyval instead
    s = sp.polyval(polinomio,raices)#evaluamos el polinomio en las raices

```



Pandas

Es una librería utilizada principalmente para manipulación y análisis de datos. El análisis de datos es un proceso que consiste en inspeccionar, limpiar y transformar datos con el objetivo de poder utilizarlos para las distintas tareas de nuestro interés. Pandas cuenta con dos estructuras para trabajar con los datos: `pandas.Series` y `pandas.DataFrame`. Las `pandas.Series` son un arreglo matricial con n elementos del mismo tipo. Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque sí su contenido. Por otro lado, las `pandas.DataFrame` es una colección de m `pandas.Series` no necesariamente del mismo tipo, pero sí con el mismo número n de elementos.

Lo primero que debemos hacer es importar la librería.

```
import pandas as pd
```

Para crear una `pandas.Series` podemos utilizar el siguiente código similar a un array de numpy:

```
In [82]: import pandas as pd

my_series = pd.Series([i*2 for i in range(5)])
my_series
```

```
Out[82]: 0    0
         1    2
         2    4
         3    6
         4    8
dtype: int64
```

También podemos crearla como si fuese un diccionario:


```
In [86]: x = pd.Series({'Ana': 10.0, 'Luis': 23.5, 'Juan': 40})
x
```

```
Out[86]: Ana      10.0
Luis      23.5
Juan      40.0
dtype: float64
```

Cada columna dentro de un dataframe es una serie. Por lo tanto los dataframes son conjuntos de series. Podemos crear un `pandas.DataFrame` de varias maneras, una de las más convenientes es utilizando diccionarios:

```
In [131]: dicc = {
            "nombre" : ["Juan", "Ana", "Luis"],
            "edad" : [20, 25, 43],
            "estatura" : [1.80, 1.62, 1.71]
          }

df = pd.DataFrame(dicc)
df
```

```
Out[131]:
```

	nombre	edad	estatura
0	Juan	20	1.80
1	Ana	25	1.62
2	Luis	43	1.71

Como se puede notar las columnas son diccionarios:

```
In [132]: df['edad']
```

```
Out[132]: 0      20
1      25
2      43
Name: edad, dtype: int64
```

Propiedades

Es posible seleccionar datos específicos dentro de nuestro dataframe utilizando `.loc` :

```
In [133]: # loc te permite seleccionar filas y columnas
df.loc[2, 'edad']
```

```
Out[133]: 43
```

```
In [134]: # También podemos seleccionar por posición
df.iloc[2, 1]
```

```
Out[134]: 43
```

Podemos consultar las propiedades de nuestros datos utilizando la siguiente función

```
In [135]: df.dtypes
```

```
Out[135]: nombre      object
          edad        int64
          estatura    float64
          dtype: object
```

Y si quisieramos cambiar algún tipo de datos también lo podemos hacer:

```
In [136]: df['edad'].astype(np.float64)
```

```
Out[136]: 0    20.0
          1    25.0
          2    43.0
          Name: edad, dtype: float64
```

Podemos conocer la forma de nuestro dataframe, es decir cuántas filas y columnas tiene y obtener el nombre de cada una de nuestra variables (columnas).

```
In [137]: print(df.shape)
          print()
          print(df.columns)
```

```
(3, 3)
```

```
Index(['nombre', 'edad', 'estatura'], dtype='object')
```

Seleccionar datos

Podemos seleccionar una o varias columnas de nuestro dataframe. También podemos filtrar datos para quedarnos únicamente con los que son objeto de nuestro estudio.

```
In [138]: #para seleccionar una columna
          df[['nombre']]
```

```
Out[138]:
```

	nombre
0	Juan
1	Ana
2	Luis

```
In [139]: #para seleccionar varias columna
df[['nombre', 'edad']]
```

```
Out[139]:
```

	nombre	edad
0	Juan	20
1	Ana	25
2	Luis	43

Podemos utilizar la sintaxis de las listas de Python para seleccionar algunos elementos de nuestro dataframe.

```
In [140]: #seleccionamos los primeros 2 elementos
df[0:2]
```

```
Out[140]:
```

	nombre	edad	estatura
0	Juan	20	1.80
1	Ana	25	1.62

```
In [141]: # seleccionar datos que cumplan con alguna de las siguientes condiciones
df[(df["edad"]>23) | (df["estatura"]<1.75)]
```

```
Out[141]:
```

	nombre	edad	estatura
1	Ana	25	1.62
2	Luis	43	1.71

```
In [142]: # seleccionar datos que cumplan con todas las siguientes condiciones
df[(df["edad"]>19) & (df["estatura"]>1.79)]
```

```
Out[142]:
```

	nombre	edad	estatura
0	Juan	20	1.8

Como vimos anteriormente podemos filtrar nuestros datos de muchas maneras y utilizando condiciones de "y"/"o".

Podemos verificar cuantos datos son nulos y si quisieramos eliminar todas estas observaciones.

```
In [143]: # Verificar por datos nulos
df.isnull().sum() #en este caso no hay nulos
```

```
Out[143]:
```

nombre	0
edad	0
estatura	0
dtype:	int64

```
In [144]: #eliminamos de esta forma todas las filas con nulos
df.dropna()
#no hay nulos así que no elimina nada
```

```
Out[144]:
```

	nombre	edad	estatura
0	Juan	20	1.80
1	Ana	25	1.62
2	Luis	43	1.71

Modificar Dataframes

Podemos agregar nuevas variables a nuestros dataframes, estas variables las podemos crear desde cero o podemos hacerlo utilizando las variables que ya estaban previamente en nuestro dataframe.

```
In [145]: # crear columna nueva de peso

df["peso"] = [100, 59, 76]

df
```

```
Out[145]:
```

	nombre	edad	estatura	peso
0	Juan	20	1.80	100
1	Ana	25	1.62	59
2	Luis	43	1.71	76

```
In [146]: # crear columna utilizando las columnas que ya tenemos

df["bmi"] = df["peso"]/df["estatura"]**2

df
```

```
Out[146]:
```

	nombre	edad	estatura	peso	bmi
0	Juan	20	1.80	100	30.864198
1	Ana	25	1.62	59	22.481329
2	Luis	43	1.71	76	25.990903

Si quisieramos agregar una nueva variable descriptiva para saber si la persona tiene sobrepeso deberiamos usar condiciones de la siguiente forma:

```
In [150]: col = 'bmi'
conditions = [ df[col]< 18.5, (df[col] <= 24.9) & (df[col]>= 18.5), df[col]
choices = ["bajo peso", 'normal', 'sobrepeso']

df["clase"] = np.select(conditions, choices, default=np.nan)
df
```

```
Out[150]:
```

	nombre	edad	estatura	peso	bmi	clase
0	Juan	20	1.80	100	30.864198	sobrepeso
1	Ana	25	1.62	59	22.481329	normal
2	Luis	43	1.71	76	25.990903	sobrepeso

Como acabamos de ver podemos manipular nuestros datos de muchas formas y generar nuevas variables utilizando condicionales y muchas herramientas que hemos visto antes.

Exportar e importar documentos

Por último, **Pandas** nos permite cargar archivos guardados localmente o que se encuentran en la web. Podemos cargar tablas previamente guardadas en formato *csv*, *json*, *parquet*, *xlsx*, etc. Lo podemos hacer de la siguiente manera:

```
import pandas as pd

datos = pd.read_csv("/path/en/donde/está/nuestro/archivo.csv")
datos2 = pd.read_json("/path/en/donde/está/nuestro/archivo.json")
```

Y también podemos exportar nuestro dataframe creado con Pandas en el formato de nuestra elección, siguiendo el siguiente código:

```
import pandas as pd

datos.to_csv("/path/en/donde/quiero/guardar/archivo.csv")
datos2.to_json("/path/en/donde/quiero/guardar/archivo.json")
```

HTML y CSS

HTML hace referencia al lenguaje de mercado que es utilizado para el desarrollo de páginas web. El código HTML nos permitirá definir el contenido que queremos incorporar a una página web, como texto, imágenes, hipervínculos, etc., en otras palabras nos permite dar estructura a nuestra página mediante etiquetas.

HTML funciona mediante etiquetas que harán referencia a los distintos elementos de nuestra página web. Un ejemplo de HTML básico se vería así.

```

<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <p>Párrafo 1</p>
  </body>
</html>

```

Cómo podemos notar siempre tendremos que iniciar nuestro código html utilizando la etiqueta `html` de apertura dentro de `<>` y tendrán que terminar con la etiqueta de clausura dentro de `</>`. Adicionalmente, el contenido de toda nuestra página web estará dentro de la etiqueta de `body`, por ejemplo, nuestro primer título se encontrará de las etiquetas `h1` y el contenido de nuestro primer párrafo estará dentro de la etiqueta `p`. Por último, el nombre que se le asignará a nuestra página web está dentro de la etiqueta `title`.

Podemos agregar en nuestra página la cantidad de títulos y párrafos que necesitemos y también podemos modificar el formato de nuestros textos.

```

<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <p>Párrafo 1.1</p>
    <p><b>Párrafo 1.2 (en negritas)</b></p>
    <p><em>Párrafo 1.3 (en cursiva)</em></p>
    <h2>Título 2</h2>
    <p><small>Párrafo 2.1 (letras más pequeñas)</small></p>
  </body>
</html>

```

Estos ejemplos sencillos nos sirven para comenzar a entender como funciona el etiquetado en html y para poder comenzar a adentrarnos a páginas con estructuras más complejas. Podemos ver que las etiquetas siempre iniciarán dentro de `<>`, dentro de las etiquetas podremos tener otros elementos como `id`, `class`, `href`, etc., que se utilizarán dependiendo el elemento que queramos incorporar.

Una manera de dar orden a nuestras páginas es dividirla por secciones o bloques. Se debe de utilizar la sintaxis de `<div>` y `</div>` para iniciar y finalizar una sección respectivamente. Todo lo que coloquemos dentro de una sección estará contenido en una especie de "caja" dentro de nuestra página web.

```

<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <div>
      <h1>Título 1</h1>
      <p>Párrafo 1</p>
      <h2>Título 1</h2>
      <p>Párrafo 2</p>
    </div>
  </body>
</html>

```

En el ejemplo anterior tanto el título 1 como el título 2 y los párrafos estarán ubicados dentro de la misma "caja".

Por último si quisieramos agregar imágenes o links a nuestra página web lo podemos hacer con sus etiquetas correspondientes de la siguiente manera. La etiqueta para imágenes es `img` y es necesario agregar adicionalmente la url asociada a la imagen que se quiere insertar.

```

<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <p>Párrafo 1</p>
    
  </body>
</html>

```

Para agregar un link es necesario utilizar la etiqueta `a` e insertar el link en el atributo `href`.

```

<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <p>Párrafo 1</p>
    <a href="https://www.itam.mx/">Página ITAM</a>
  </body>
</html>

```

Todos los ejemplos que aquí se muestran se pueden utilizar con un editor de texto y guardando los archivos como .html.

CSS

CSS es un lenguaje que trabaja en conjunto con HTML y cuya finalidad es darle el estilo deseado a los elementos de nuestras páginas. En CSS se establecen las instrucciones para que nuestra información se muestre con el formato que queramos. Podremos pasarle formatos a clases, ids o incluso divisiones completas utilizando un script independiente a nuestro html con un lenguaje similar a los diccionarios de Python donde indicaremos las etiquetas a las que queremos modificar y el formato que se les dará.

Por lo tanto, si tuviéramos el siguiente script de html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>página</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <p id="parrafo1"> Párrafo 1</p>
    <span class="clase1"> Span 1 </span>
  </body>
</html>
```

y quisieramos cambiar el color en toda nuestra página web a un color azul utilizaríamos el siguiente script de css:

```
body {
  background-color: #F0F8FF;
}
```

y quisieramos cambiar el formato de alguna clase o id específico del html anterior, utilizaríamos el siguiente script:

```
#parrafo1{
  color: #0000FF
}

.clase1{
```

XPath

XPath es un lenguaje que nos permite construir expresiones para extraer información de un documento XML, como los html. XPath permite buscar y seleccionar los elementos que nos interesan teniendo en cuenta la estructura jerárquica de XML.

Para realizar búsquedas en XPath es necesario conocer algunas particularidades de este lenguaje.

Primero se debe definir el prefijo de búsqueda:

- `//` : se utiliza para realizar búsquedas en cualquier nivel del documento.
- `/` : se utiliza para realizar búsquedas en la raíz del documento.
- `./` : se utiliza para realizar la búsqueda de forma relativa.

Posteriormente, debemos indicar el nodo al que nos queremos dirigir utilizando el tag. Después podemos utilizar predicados para focalizar más la búsqueda, por ejemplo:

```
//tag[predicado]
```

Podemos hacer consultas utilizando el lenguaje XPath desde Python de la siguiente manera:

Es importante notar que para todas las tareas de webscraping será necesario importar la librería `requests`, la cual se encargará de brindarnos el dom de las pags web y con ello podremos obtener la información que ahí se encuentre.

```
import requests
```

```
In [6]: #Ejemplo: obtener la misión del ITAM desde su página web

import requests
from lxml import html

url = "https://www.itam.mx/"

respuesta = requests.get(url) #hacer el requerimiento del servidor

parser = html.fromstring(respuesta.text) #parsear nuestro texto

parser.xpath("//div[@class='cita-ITAM']/p/text()")[1]
```

```
Out[6]: 'Contribuir a la formación integral de la persona humana y al desarrollo
de una sociedad más libre, más justa y más próspera.'
```

BeautifulSoup

Beautiful Soup es una librería de Python para analizar documentos HTML, la cual con el dom creará un arbol con todos los elementos de la página de nuestro interés para extraer información.

Es una herramienta muy útil y fácil de utilizar para realizar webscraping en páginas complejas.

Para entender cómo funciona se puede consultar el siguiente ejemplo donde generamos una función para extraer las noticias de un periódico y generar un dataframe con los datos de nuestro interés:


```

In [10]: from bs4 import BeautifulSoup
from datetime import date
import datetime
import time
import pandas as pd

link = 'https://www.economy.com.bo' #Link de la página de noticias
#lista de todas las secciones
secciones = ["business", "life"]

#obtenemos urls de las secciones.
link_pages = []
for seccion in secciones:
    for i in range(1,3): #buscamos página en todas las páginas de la sección
        link_pages.append(f'https://www.economy.com.bo/blog/section/{seccion}-{i}')

#obtenemos los urls de cada uno de los artículos que se encuentren en las secciones
link_articles = []
for i in link_pages:
    r = requests.get(i)
    soup = BeautifulSoup(r.content, 'html.parser')

    for j in soup.findAll("h2", {"class": "title"}):
        try:
            link_a = j.find('a').get('href')
            link_complete = f"{link}{link_a}"
            if link_complete not in link_articles:
                link_articles.append(link_complete)
        except:
            pass

def economy(link_articles):
    autores = []
    cuerpos = []
    titulos = []

    for link in link_articles:
        r = requests.get(link) #hacer el requerimiento del servidor
        soup = BeautifulSoup(r.content, 'html.parser') #parseamos

        author = soup.find('div', class_='author-information') #buscamos el autor
        cuerpo = soup.find_all('div', class_='content-body') #buscamos el cuerpo
        title = soup.find('h1', class_='title') #buscamos el título en el cuerpo

        if cuerpo == None : #Cuando no encuentra nada sustituimos por un valor nan
            cuerpos.append("nan") #algunos articulos son varias imagenes
        else:
            cuerpos.append(cuerpo[0].text) #agregamos el texto de la noticia

        if author == None :
            autores.append("nan")
        else:
            autores.append(author.text.replace("\n", "")) #guardamos el autor

```

```
    if title == None :
        titulos.append("nan")
    else:
        titulos.append(title.text.replace("\n", "")) #guardamos el

#creamos un dataframe con los datos que recolectamos
df = pd.DataFrame({
    'link_article': link_articles,
    'author': autores,
    'title': titulos,
    'text': cuerpos })

return(df)

economy(link_articles) #corremos la función
```

Out[10]:

	link_article	author	title	text
0	https://www.economy.com.bo/articulo/business/e...	Economy.com.bo	Entel habilita paquetes dirigidos al teletraba...	\n\nLos usuarios del servicio de Telefonía Móv...
1	https://www.economy.com.bo/articulo/business/m...	Economy.com.bo	Marcelo Claire, director de operaciones de Sof...	\n\n\n\nEl director de operaciones de SoftBank...
2	https://www.economy.com.bo/articulo/business/t...	Fernando Aguirre	Tul, la startup colombiana que proyecta expand...	\n\nEn 2 años de vida, la startup colombiana T...
3	https://www.economy.com.bo/articulo/business/j...	nan	Johnson & Johnson estima \$us 3.500 millones en...	\n\nJohnson & Johnson estimó el martes ingreso...
4	https://www.economy.com.bo/articulo/business/m...	Economy.com.bo	Minkedu es seleccionada para el programa Build...	\n\nStart-Up Chile una de las aceleradoras par...
...
75	https://www.economy.com.bo/articulo/life/arte-...	Cristina Chilo Crespo	Arte boliviano que vale oro	\n\nA escala mundial, existen obras de arte qu...
76	https://www.economy.com.bo/articulo/life/grani...	nan	Granizada y lluvia causan inundaciones en Tarija	\n\nLa tarde de ayer se reportó una intensa ll...
77	https://www.economy.com.bo/articulo/life/lider...	nan	Líderes mundiales se reúnen en la cumbre contr...	\n\nLíderes del mundo se reúnen este lunes en ...
78	https://www.economy.com.bo/articulo/life/mundo...	nan	El mundo llega a la COP26 con la producción e...	\n\nLos países proyectan un aumento en la gene...
79	https://www.economy.com.bo/articulo/life/reali...	Economy.com.bo	Realizan con éxito el primer Zumbathon Rosa co...	\n\nLa Secretaría Municipal de Salud de La Paz...

80 rows × 4 columns

Selenium

Selenium es una herramienta que nos puede servir como complemento para el webscraping. Si bien es cierto que la mayoría de las páginas las podemos scrapear usando únicamente BeautifulSoup Habrán ocasiones donde las páginas sean dinámicas, es decir que necesiten de

interacción para mostrar algunos elementos. Un ejemplo es Google, si quisiéramos encontrar todos los links que aparecen cuando se busca una palabra al azar tendríamos que escribir dicha palabra en la barra buscadora y presionar enter. Selenium nos ayudará a realizar todas estas acciones que hacemos día a día y que a veces son necesarias para mostrar información, como dar clicks, escribir texto, presionar enter en nuestro teclado, etc.

Podemos ver algunos ejemplos sencillos de cómo encontrar y presionar el primer link de la primera página de Google al buscar la palabra "itam" y de ahí traer la misión una vez más combinando todas las herramientas vistas requests, selenium y BeautifulSoup.

```
In [21]: from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

s=Service(ChromeDriverManager().install()) #instalamos el driver
driver = webdriver.Chrome(service=s)
driver.maximize_window()
driver.get('https://www.google.com') #obtenemos el driver
#ya tenemos acceso a todos los elementos de la página
driver.find_element(By.NAME, 'q').send_keys('itam',Keys.ENTER)#buscamos la

driver.find_element_by_class_name("yuRUbf").find_element_by_tag_name("a").c

link_itam = driver.current_url #nos regresa el link de la página en la que

#utilizamos las herramientas que ya habíamos visto
r = requests.get(link_itam)
soup = BeautifulSoup(r.content, 'html.parser')

soup.find('div', class_='cita-ITAM').find_all('p')[1].text
```

```
===== WebDriver manager =====
Current google-chrome version is 98.0.4758
Get LATEST chromedriver version for 98.0.4758 google-chrome
Driver [/Users/joaquinbarrutia/.wdm/drivers/chromedriver/mac64/98.0.4758.
48/chromedriver] found in cache
/var/folders/l_/pc12z1g16z359gcfw1jgy3d00000gn/T/ipykernel_1247/403002661
8.py:13: DeprecationWarning: find_element_by_* commands are deprecated. P
lease use find_element() instead
  driver.find_element_by_class_name("yuRUbf").find_element_by_tag_name
("a").click()
```

```
Out[21]: 'Contribuir a la formación integral de la persona humana y al desarrollo
de una sociedad más libre, más justa y más próspera.'
```

Regex

Las expresiones regulares nos sirven para encontrar patrones dentro de los textops y de esta forma seleccionar las partes que nos interesan o limpiarlo de acuerdo a nuestras necesidades.

En python se utiliza la librería de `re` .

```
import re
```

Podemos verificar algunos de los elementos del glosario de las expresiones regulares mediante un ejemplo:

```
In [104]: import re
```

```
texto = "Pedro Pérez Pita pintor perpetuo,\n        pinta paisajes por el poco precio \n        para poder partir \n        pronto para París."
```

```
In [30]: #buscamos una palabra específica\nre.search("pintor",texto)
```

```
Out[30]: <re.Match object; span=(17, 23), match='pintor'>
```

```
In [31]: texto[17:23]
```

```
Out[31]: 'pintor'
```

```
In [99]: re.findall("\w*P\w+",texto) #encuentra todas las palabras que empiezan con
```

```
Out[99]: ['Pedro', 'Pérez', 'Pita', 'París']
```

```
In [121]: re.findall("el (\w+)",texto) #encontramos palabra despues de el
```

```
Out[121]: ['poco']
```

```
In [122]: re.sub("París","Perú ",texto) #podemos realizar sustituciones
```

```
Out[122]: 'Pedro Pérez Pita pintor perpetuo,pinta paisajes por el poco precio para\n        la poder partir pronto para Perú .'
```

Procesamiento de Lenguaje Natural

El procesamiento del lenguaje natural (NLP) es una rama de la Inteligencia Artificial que estudia la interacción del lenguaje humano y las computadoras. De manera que por medio de patrones la computadora pueda entender, predecir y hallar elementos del texto.

Podemos ver algunas aplicaciones a continuación.

Lo primero que se debe hacer es tokenizar nuestro texto, es decir dividirlo por palabras o por oraciones pequeñas, hay debates sobre como tokenizar, ya que palabras como New York si son tokenizadas por palabra no harían el mismo sentido.

```
In [128]: import nltk
import urllib.request
```

```
In [131]: response = urllib.request.urlopen('https://es.wikipedia.org/wiki/M%C3%A9xic
html = response.read()

soup = BeautifulSoup(html,"html5lib") #limpia el texto

text = soup.get_text(strip=True)

print (text)
```

...

```
In [132]: from nltk.tokenize import word_tokenize

tokens = word_tokenize(text,"spanish") #tokens en español
tokens=[word.lower() for word in tokens if word.isalpha()] #remover los sig
print(tokens)
```

...

```
In [133]: #verificamos con que frecuencia aparece cada palabra

freq = nltk.FreqDist(tokens)
freq.items()
```

...

Posteriormente, para un mejor analisis es conveniente eliminar las stop words que son palabras que si son eliminadas aún se puede entender el texto.

```
In [134]: from nltk.corpus import stopwords

clean_tokens = tokens[:]

for token in tokens:

    if token in stopwords.words('spanish'):

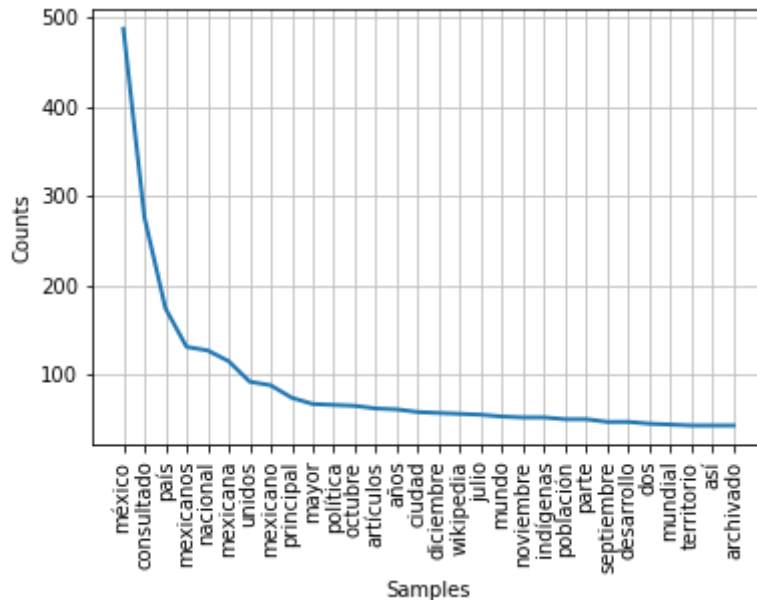
        clean_tokens.remove(token)
```



```
In [135]: #volvemos a calcular la frecuencia con las palabras relevantes
freq = nltk.FreqDist(clean_tokens)
for key,val in freq.items():
    print (str(key) + ':' + str(val))
```

```
méxico:487
wikipedia:56
enciclopedia:2
wgbreakframes:1
false:9
wgseparatortransformtable:1
wgdigittransformtable:1
wgdefaultdateformat:1
dmy:1
wgmonthnames:1
enero:35
febrero:15
marzo:20
abril:25
mayo:30
junio:29
julio:55
agosto:26
septiembre:47
```

```
In [136]: # visualizamos la frecuencia
freq.plot(30, cumulative=False)
```



```
Out[136]: <AxesSubplot:xlabel='Samples', ylabel='Counts'>
```