



UNIVERSIDAD
DE CHILE

Manual de usuario programas de simulación y modelación

Documento creado como complemento a la tesis “Uso de Python para la modelación y determinación de vida útil de matrices alimentarias mediante simulación por Montecarlo” por Joseph Huequelef de la Universidad de Chile (año 2023).

Acerca de este manual

El programa de simulación busca acercar una simulación estadística al público en general que tenga poco o nada de conocimiento sobre el método de Montecarlo y las modelaciones de datos.

El programa se desarrolló siguiendo la misma lógica utilizada por Escobedo y Velázquez en su trabajo “inclusion of the variability of model parameters on shelf-life estimations for low and intermediate moisture vegetables” (2012), solo a que diferencia de ellos, se desarrolló una programación en Python para obtener los mismos resultados para el caso de la simulación por Montecarlo para la determinación de vida útil.

Para los programas de modelamiento se utilizaron bibliotecas y recursos disponibles para la modelación de datos de tendencia no lineal.

Los programas entregan resultados y gráficos según se va ejecutando.

Los resultados pueden variar según los datos ingresados (esto también es válido para la modelación de datos).

Este manual no garantiza aprender la lógica de la programación, ya que esto se logra interactuando y creando distintos tipos de programas.

Índice

instalación de Python e interprete	4
bibliotecas utilizadas	7
operadores básicos de Python	7
operadores relacionales	8
Primeros pasos y palabras reservadas del lenguaje.....	9
Declaración de variables	10
Colección de elementos	13
Operaciones con listas	15
Recorriendo una lista	17
Ciclo while.....	20
Otros objetos en donde almacenar elementos e información.....	21
Recibir datos por parte del usuario.....	23
Programa1	26
Programa2.....	27
Programa3.....	28
Lectura y escritura de archivos	31
Dataframes.....	33
Dataframes con listas.....	35
Matrices.....	37
Crear matrices con numpy.....	40
Aplicación de matrices en la simulación de Montecarlo.....	43
Uso del programa	46
Modelamiento de datos.....	53
Referencias.....	58

instalación de Python e interprete


Antes de iniciar y explicar los programas se dará a conocer al lector algunos elementos básicos en la programación en Python. La idea es que el usuario vaya ejecutando las líneas de código que se van apareciendo.

Antes que todo, lo primero es instalar Python desde la página oficial de Python, para realizar esta acción se puede acceder al paquete de instalación siguiendo la siguiente dirección de enlace: <https://www.python.org/downloads/>, en donde automáticamente se seleccionará la versión compatible con nuestro sistema operativo, en caso de que esto no ocurra, se debe navegar por la misma página hasta encontrarlo. Una vez pinchado el enlace, se debería ver una ventana como aparece a continuación (imagen1).



Imagen 1

Luego que se sigan los pasos de instalación, se instala el intérprete de Python. Este intérprete se encargará de traducir las líneas de código escritas en el a lenguaje máquina para así mostrar resultados por pantalla. Recomendando instalar “Anaconda” ya que contiene todos los paquetes para iniciarse en el mundo de la “ciencia de datos”. Dentro de todos estos paquetes se trabajará con el módulo “Spyder” (imagen 3) para realizar la programación, para instala Anaconda y sus paquetes se puede ir al siguiente enlace <https://www.anaconda.com/download> y se visualizará esta imagen (imagen2).


ANACONDA

[Enterprise](#)
[Pricing](#)
[Resources](#)
[About](#)

Anaconda Distribution

Free Download

Everything you need to get started in data science on your workstation.

- ✓ Free distribution install
- ✓ Thousands of the most fundamental DS, AI, and ML packages
- ✓ Manage packages and environments from desktop application
- ✓ Deploy across hardware and software platforms

Imagen2







 <p>DataSpell</p> <p>DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.</p> <p>Install</p>	 <p>CMD.exe Prompt</p> <p>0.1.1</p> <p>Run a cmd.exe terminal with your current environment from Navigator activated</p> <p>Launch</p>	 <p>JupyterLab</p> <p>3.5.3</p> <p>An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.</p> <p>Launch</p>
 <p>Qt Console</p> <p>5.4.0</p> <p>PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.</p> <p>Launch</p>	 <p>Spyder</p> <p>5.4.1</p> <p>Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features</p> <p>Launch</p>	 <p>VS Code</p> <p>1.77.3</p> <p>Streamlined code editor with support for development operations like debugging, task running and version control.</p> <p>Launch</p>

Imagen 3 Dentro de anaconda se encuentran varios módulos, dentro de ellos Spyder. Si anaconda no da opción de abrir el programa, se debe instalar el paquete desde la misma interfaz anaconda.

Instalación de bibliotecas en spyder

Las bibliotecas son colecciones de funciones que agilizan la programación, es decir, alguien ya escribió una función de interés (en lenguaje C) y la dejó a libre disposición para que los usuarios la ocupen y no tengan que perder tiempo programando esa parte y que podamos ocuparla directamente solo importando estas funciones. De estas bibliotecas podemos ocupar todos los elementos de la biblioteca o algunos específicos, esto varía según lo que se requiera programar.

Para hacer uso de las bibliotecas, lo primero que se debe hacer es instalarlas. Esto se logra abriendo la “Powershell Prompt” la cual se encuentra en anaconda. Al abrir la PowerShell prompt se debe ver esto (imagen 4).

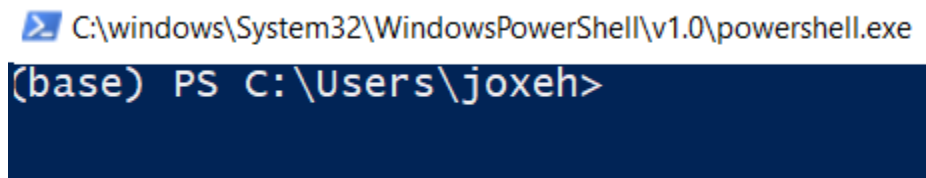


Imagen 4

En esta interfaz se escriben las líneas de código para la instalación de las bibliotecas. A modo de ejemplo se instala la biblioteca “math”.

Para eso se escribe el comando “pip”, palabra reservada para la instalación de módulos en Python antes de escribir el nombre de la biblioteca que se desea instalar (imagen 5).

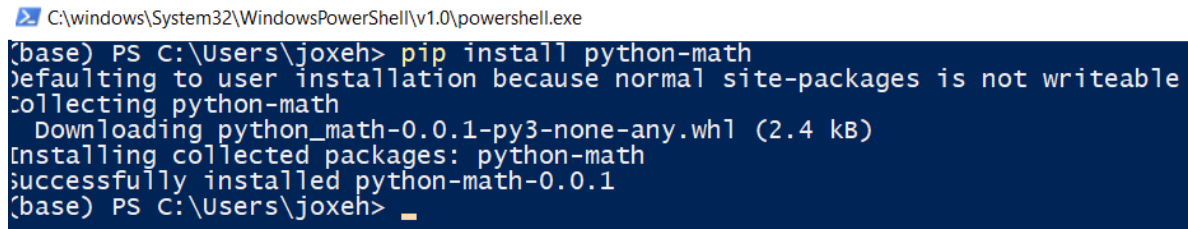


Imagen5. Instalación de la biblioteca “math” en anaconda PowerShell.

Existen muchas bibliotecas para utilizar. Estas bibliotecas se instalan según los requerimientos del programador. Las líneas de comando pueden variar según la biblioteca a instalar pero por lo general se escribe el nombre de la biblioteca después de escribir pip. En caso de que se requiera, en internet existen varios recursos para consulta de instalación de diversas bibliotecas.

Bibliotecas más utilizadas

Pandas: Su página web la define como una herramienta de manipulación y análisis de datos de código abierto, rápida, potente, flexible y fácil de usar construida en el lenguaje de programación Python.

Numpy: Se define a sí misma como una biblioteca fundamental para la informática científica en Python. Es excelente para trabajar con matrices ya que es capaz de crear un objeto de tipo matriz según los requerimientos del programador. Numpy contiene matrices multidimensionales y estructuras de datos matriciales.

Matplotlib: Es una biblioteca creada para la presentación de información visual, como gráficos e imágenes de manera interactiva. Las imágenes generadas se pueden modificar para mostrar información de interés o resaltar resultados.

Operadores básicos en Python

+ : Sirve para sumar y concatenar elementos (variables)

-: se utiliza para realizar la operación resta entre en números

*****: Representa a la operación de multiplicación.

******: Representa la operación potencia en Python

/: representa la división de números.

%: Obtiene el módulo o resto de la división.

//: Realiza la operación de división obteniendo solo la parte entera de la operación.

=: operador de asignación. Asigna un valor a una variable.

Operadores lógicos

And = Devuelve verdadero si ambos valores son verdaderos

Or = devuelve verdadero si al menos uno del operando es verdadero

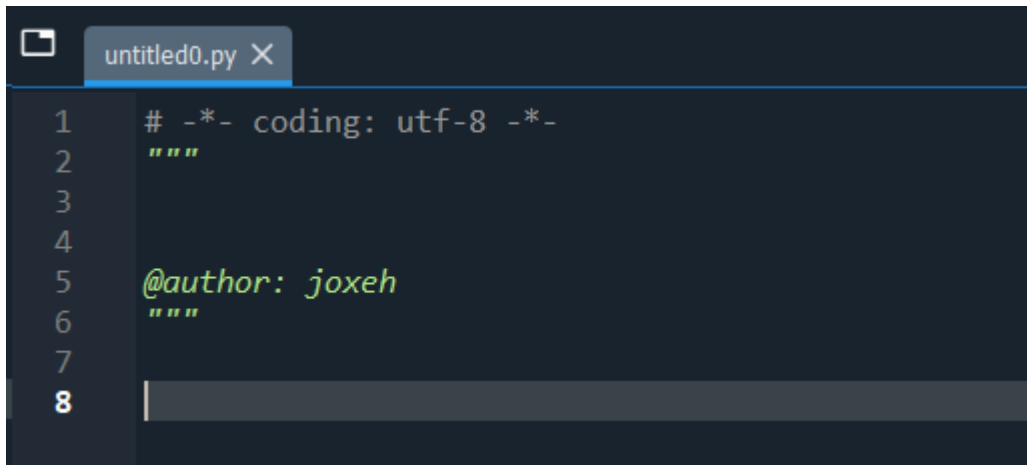
Not: Devuelve verdadero si alguno de los elementos es falso.

Operadores relacionales

- <: Devuelve verdadero si el operando de la derecha es mayor al de la izquierda
- >: Devuelve verdadero si el operando de la izquierda es mayor al de la izquierda
- <=: Devuelve verdadero si el operando de la derecha es mayor o igual al de la izquierda
- >=: Devuelve verdadero si el operando de la izquierda es mayor o igual al de la izquierda
- ==: Devuelve el valor verdadero si ambos valores son idénticos.
- !=: Devuelve verdadero si ambos operandos son distintos.

Primeros pasos y palabras reservadas del lenguaje

Ya instalado lo necesario se comenzará con la práctica de la programación en el lenguaje de programación Python. Para esto se debe abrir “Spyder” y se debe visualizar lo siguiente (imagen6).



```
1  # -*- coding: utf-8 -*-
2  """
3
4
5  @author: joxeh
6  """
7
8
```

Imagen 6 Esto se debe visualizar al abrir el módulo spyder

Luego de abrir Spyder se pueden escribir líneas de código en la línea en donde este alumbrando el cursor. Este cursor siempre estará presente en los distintos interpretes existentes en Python.

Lo primero que se hará es imprimir un mensaje por pantalla por parte del programador. Esto se logra usando la palabra reservada “Print () (imagen 7)”. Una palabra reservada en Python es una función que no puede ser utilizada para definir o nombrar una variable.

Al escribir una frase alfanumérica o un numero dentro de Print arrojará un mensaje por pantalla (imagen 8).

```
untitled0.py X
1  # -*- coding: utf-8 -*-
2  """
3
4
5  @author: joxeh
6  """
7
8  print("Lea el manual por favor!")
```

```
In [2]: runfile('C:/Users/joxeh/Desktop/postgrado/untitled0.py',
postgrado')
Lea el manual por favor!
```

Imagen 8

Con esta línea de código se puede crear un programa sencillo, el cual es capaz de mostrar cualquier mensaje que se escriba dentro del Print ().

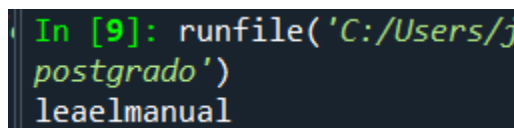
Declaración de variables

Ahora que ya sabemos ocupar la función Print (), empecemos a realizar experimentos con Python. Que ocurre si separamos la frase anterior en varias partes? y que acá parte la separamos en distintas variables? prestemos atención a la siguiente imagen (imagen9).

```
5  @author: joxeh
6  """
7
8  variable1 = "lea"
9  variable2 = "el "
10 variable3 = "manual"
11 print(variable1+variable2+variable3)
12
```

Imagen9

Si imprimimos la suma de estas 3 variables se obtiene el siguiente mensaje por pantalla imagen 10.



```
In [9]: runfile('C:/Users/j  
postgrado')  
leaelmanual
```

Imagen 10

El resultado final fue la misma frase, pero todo junto!. Esto es debido a que Python concatena (junto) todas las variables en una sola. Esto se llevó a cabo con la operación “+”.

Si se busca mostrar el mismo mensaje originalmente se debe agregar un “espacio” antes o después de las variables. Los espacios también son considerados caracteres en los lenguajes de programación.

La concatenación de elementos debe ser del mismo tipo, es decir, texto con texto y números con números. Los elementos que no son numéricos se denominan datos “String”. Los datos String se pueden considerar como texto, palabras, letras o también números ingresados entre comillas, los cuales son interpretados como String. Estos valores tipo String numéricos se pueden convertir en números si se ocupa la función int o float por ejemplo:

Variable = “85”

La cual devolverá por pantalla si imprimimos variable:

“85”

Se indica que variable es de tipo String, si embargo si ocupamos por ejemplo:

Int(variable) y se imprime, esto devolverá por pantalla el siguiente valor numérico:

85

Como se aprecia, paso de ser un String a una variable de tipo numérica.

Se pueden realizar operaciones con String pero no se pueden “sumar” texto con números, es decir, no se puede mezclar peras con manzanas. Si se llega a realizar esto, Python arrojará un error como en el siguiente ejemplo imagen 10. Obviamente si se suman dos variables de tipo numérica estos darán por resultado la suma de ambas variables como se hace normalmente en la matemática.

```
5  @author: joxeh
6  """
7
8  variable_string = "manual"
9  variable_numerica = 5
10 suma = variable_string + variable_numerica
11 print(suma)
12
```

Imagen 11

```
File c:\users\joxeh\desktop\postgrado\untitled0.py:10
    suma = variable_string+variable_numerica
TypeError: can only concatenate str (not "int") to str
```

Imagen 12

Ejemplo de error producido por intentar concatenar una variable de tipo String y una variable numérica.

Si se realiza una operación de multiplicación a la variable_string, esta arrojará lo que contiene la variable pero multiplicada por la cantidad indicada. Si se intenta dividir por un número, esto arrojará error.

Colección de elementos

A una colección de elementos se le conoce comúnmente como “lista” y es un objeto de tipo “list”. Las listas se declaran utilizando dos corchetes vacíos ([]) después del nombre que se le quiera dar.

En Python todo es un objeto y varían en características y propiedades. Un String es un objeto de tipo “Str”(String) mientras que un objeto numérico, puede ser de carácter entero (int), decimal (float) o complejos (complex). Por lo general los datos numéricos más comunes son los de tipo entero y flotante. Observemos el siguiente ejemplo de colección de elementos (imagen 11).

```
4
5  @author: joxeh
6  """
7
8  lista = ["casa",3,"lentes",3.14,"#"]
9  print(lista)
10
```

Imagen 13

La imagen anterior muestra una lista llamada “lista” la cual contiene 5 elementos, 3 elementos de tipo String (por las comas que lo rodean) y dos numéricas.

Si se imprime por pantalla la lista se obtiene lo siguiente.

```
In [14]: runfile('C:/Users/joxeh/D
postgrado')
['casa', 3, 'lentes', 3.14, '#']
```

Imagen 14

En donde los corchetes indican que se está en presencia de una colección de elementos.

Las listas son elementos mutables es decir, se pueden modificar agregando elementos, eliminando elementos, multiplicarlos por un escalar, concatenarlas con otras listas, y pueden tener elementos repetidos.

Para saber qué tipo de objeto es la variable lista, se puede utilizar la función “type” la cual sirve para consultar a que tipo de objeto pertenece la variable de la siguiente forma:

```
print(type(lista))
```

Esto mostrara un mensaje por pantalla indicando que la variable lista pertenece a los objetos de tipo list.

Una lista puede tener cientos de millones de elementos, de cierta forma solo se está limitado por la memoria de nuestra computadora.

Cada elemento de la lista está asociada a un índice según su orden, el primer elemento de una lista siempre ocupa la posición 0 y no la posición 1. Entender esto es fundamental a la hora de trabajar con listas, ya que habrá ocasiones en donde necesitaremos un elemento específico de la lista y debemos saber dónde buscarlo.

Para el caso del ejemplo anterior “casa” ocupa la primera posición y por eso su índice asociado es el 0, 3 ocupa la segunda posición y su índice corresponde al 1, “lentes” ocupa la tercera posición y le corresponde el índice 2, 3.14 ocupa la cuarta posición y le corresponde el índice 3, “#” ocupa la quinta posición y le corresponde el índice 4 (imagen 12).

Generalizando los índices van desde 0 hasta n-1.

Vector	“casa”	3	“lentes”	3.14	“#”
posición	0	1	2	3	4

(imagen 15)

Operaciones con listas

Las listas permiten que se hagan un gran numero de operaciones con ellas. Algunas de estas operaciones (métodos) disponibles son:

Append: Permite agregar un elemento al final de la lista. Por ejemplo volvamos a la lista inicial.

Lista = [“casa”, 3, “lentes”, 3.14, “#”]

Si se desea añadir un elemento por ejemplo el elemento String “libro”. Para llevar esto a cabo solo se debe seguir el siguiente procedimiento.

lista.append(“libro”)

Lo que devolverá al imprimir la lista por pantalla.

Lista = [“casa”, 3, “lentes”, 3.14, “#”, “libro”]

Como se aprecia ahora la lista inicial paso de tener 5 a 6 elementos.

Insert: Permite añadir un elemento en la posición que se desee. Supongamos que queremos añadir el elemento “plumón” en la primera posición. para esto se debe utilizar la función insert de la siguiente forma.

```
Lista.insert(0, "plumón")
```

En donde el numero 0 indica el índice o posición en el que se quiere añadir el elemento, es este caso es la primera posición (índice 0) y en segundo lugar el elemento a añadir. Ahora la lista queda de la siguiente forma.

```
Lista = ["plumon", "casa", 3, "lentes", 3.14, "#", "libro"]
```

Ahora la lista tiene 7 elementos y el elemento “plumón” quedo como primer elemento.

del: De forma análoga se puede eliminar un elemento en particular siguiendo la misma sintaxis que el ejemplo anterior. Supongamos que queremos eliminar el elemento “casa” el cual ocupa la posición 1, para esto se debe escribir:

```
del lista [1]
```

devolviendo

```
Lista = ["plumón", 3, "lentes", 3.14, "#", "libro"]
```

También se pueden eliminar elementos de forma simultánea indicando un índice que indique el principio y otro que indique el final. Por ejemplo se toma la lista modificada anteriormente y se quieren eliminar los siguientes elementos,

```
del lista [0:2]
```

Esto quiere decir que se desean eliminar elementos de la posición 0 hasta la posición 1, esto debido a que los índices de las listas van desde 0 hasta n-1 como se mencionó anteriormente. Lo que devolverá:

```
Lista = [ "lentes", 3.14, "#", "libro"]
```

Devolviendo una lista que solo tiene 4 elementos si es que se aplica la función Print a la variable lista modificada.

Si no se añaden los índices, por defecto Python seleccionara los elementos desde el primero hasta el último.

```
del lista [:]
```

Para este caso se eliminará todos los elementos de la lista y esta quedará vacía.

Remove: elimina un elemento en particular sin indicar el índice. En caso de que el elemento este repetido en la lista, solo se eliminara el primer elemento, por ejemplo:

```
lista.remove("lentes")
```

devolverá lo siguiente

```
Lista = [ 3.14, "#", "libro"]
```

Pop: También sirve para eliminar elementos por índices, si no se le indica el índice eliminara el ultimo valor de la lista. Si se aplica la función Print junto a la función pop, mostrara por pantalla el elemento eliminado.

```
Lista = [ 3.14, "#", "libro"]
```

```
Print (lista.pop ())
```

```
"libro"
```

Reverse: invierte el orden de los elementos de la lista, por ejemplo:

```
Lista.reverse()
```

Devuelve

```
Lista = ["libro", "#", 3.14]
```

Supongamos que tenemos la siguiente lista de elementos:

```
Lista_2 = [2 ,5 ,9 ,7 ,1]
```

Una lista de carácter numérica que tiene 6 elementos.

Si la lista es muy voluminosa y se quiere conocer su longitud esto se logra mediante la función:

Len: Esta función devuelve la longitud o tamaño del vector, lo que al aplicar esta función en la lista_2, esta devolverá un valor numérico igual a 6.

```
Print(len(lista_2))
```

Ejemplo de la aplicación len la cual devuelve el valor de 6.

Max: devuelve el mayor valor dentro de la lista, ejemplo

```
Print(max(lista_2))
```

Esto devuelve el valor de 9, el cual es el valor máximo

Min: Devuelve el valor mínimo de una lista. La sintaxis es la misma solo que en vez de escribir max, de debe escribir min. Esto devuelve el valor numérico de 1.

Sum: Devuelve el valor correspondiente al valor numérico de la suma de todos los elementos de la lista. Todos los elementos deben ser numéricos, ya que como se mencionó anteriormente, si se intenta sumar una variable numérica con una de tipo de String esto devolverá un error. Para este caso al utilizar la función sum, dará como resultado un valor numérico igual 24.

```
Print(sum(lista_2))
```

In: devuelve verdadero falso según sea el caso. Devolverá verdadero si el elemento buscado se encuentra dentro de la lista, o falso si es que este elemento no se encuentra. Para buscar un elemento se debe hacer de la siguiente forma.

```
Elemento in lista
```

Para el caso de nuestra lista, le preguntare a Python si el numero -1 está en la lista y que me devuelva por pantalla si es verdadero o falso de la siguiente forma:

```
Print (-1 in lista)
```

Acción que devolverá falso porque -1 no está contenido en la lista_2.

Estas son algunas de las operaciones que se pueden hacer a una lista

Recorriendo una lista

Como se mencionó anteriormente una colección de elementos corresponde a una lista, y las listas pueden modificarse y también recorrerse. Por lo general, para recorrer una se ocupa el ciclo “for”. El ciclo for y el ciclo while son funciones utilizadas para iterar (repetir muchas veces una operación) sobre una o mas listas. La diferencia entre ocupar el ciclo for y el ciclo while, es que en el ciclo while se usan condiciones para llevar a cabo las iteraciones, como las condiciones de inicio y termino así como un contador asociado.

Supongamos que tenemos la siguiente lista:

```
Lista_3 = [1,2,3,4,5,6,7,8,9,10]
```

Se puede imprimir todos los elementos por pantalla de la siguiente forma utilizando un ciclo for:

```
for elemento in lista_3:  
    print(elemento)
```

Llevando esto al interprete se tiene lo siguiente:

```

4
5 @author: joxeh
6 """
7
8 lista_3 = [1,2,3,4,5,6,7,8,9,10]
9
10 for elemento in lista_3:
11     print(elemento)

```

```

In [6]: runfile('C:/User
postgrado')
1
2
3
4
5
6
7
8
9
10

```

Imagen 16

Como se puede apreciar se imprimió todos los valores contenidos en la lista. La instrucción se puede pensar de la siguiente manera, “para los elementos contenidos en la lista, quiero que se impriman los elementos hasta que todos aparezcan por pantalla”

El ciclo for irá recorriendo los elementos de la lista uno por uno y por cada uno que encuentre imprimirá su valor por pantalla. Los “:” deben ir en la línea del ciclo for, ya que después de esos “:” se indica la instrucción a llevar a cabo.

También se puede realizar distintos tipos de operaciones, por ejemplo, imprimiré las primeras 10 potencias de 2 de la siguiente forma (imagen 17).

```

5 @author: joxeh
6 """
7
8 lista_3 = [1,2,3,4,5,6,7,8,9,10]
9
10 for elemento in lista_3:
11     potencia = 2**elemento
12     print(potencia)
13

```

Imagen 17

Obteniendo como resultado

```
In [7]: runfile('C:/Users/jo  
postgrado')  
2  
4  
8  
16  
32  
64  
128  
256  
512  
1024
```

Imagen 18

También se pudo escribir de la siguiente manera obteniéndose el mismo resultado (imagen 19).

```
4  
5 @author: joxeh  
6 ""  
7  
8 lista_3 = [1,2,3,4,5,6,7,8,9,10]  
9  
10 for elemento in lista_3:  
11     print(2**elemento)  
12
```

Imagen 19

Ciclo While

Se puede obtener el mismo resultado con el ciclo while de la siguiente forma imagen (20):

```
5  @author: joxeh
6  ""
7
8  contador = 1
9  while contador <=10:
10     print (2**contador)
11     contador = contador+1
12
13
```

Imagen (20)

Con el ciclo while aparece un contador, el cual este encargado de subir o disminuir cierta cantidad de unidades según como este definido. Para este caso el, contador subirá una unidad (+1) luego de ejecutar la instrucción que sigue luego de los ":". En un principio el contador tiene un valor de "1" en su primera ejecución, luego en la segunda tendrá el valor de 2 y así sucesivamente hasta que alcance el valor de 10, en donde se detendrá, ya que el while indica explícitamente que se ejecutara hasta que la variable contador alcance el valor igual a 10. Si se le quita el "igual que" a la condición, el contador solo alcanzara el valor de nueve. Queda como tarea cambiar el valor del contador inicial de 1 a 0 y observar que ocurre con los datos generados, ¿sigue entregando 10 valores?.

Otros objetos en donde almacenar elementos e información

Es importante mencionar que no solo las listas se encargan de almacenar elementos, si que existen otros objetos capaces de almacenar información. Dentro de estos objetos se encuentran las "tuplas", "diccionarios" y los "set" también llamados conjuntos.

Tuplas: o tuples (em ingles), tiene algunas características parecidas a las listas en cuanto a guardar información, pero tienen una gran diferencia, las tuplas a diferencias de las listas **no pueden ser modificadas**, es decir, **no se puede agregar no eliminar elementos**. Si bien se pueden concatenar dos tuplas diferentes o multiplicarlas por un escalar, lo que se hace es generar una nueva tupla, ya que la original permanece constante. Las pociones de las tuplas siguen la misma lógica que las listas, ya que el primer elemento ocupa la posición 0.

Para declarar una tupla de utilizan paréntesis en ves de corchetes de la siguiente forma:

```
mi_tupla = (2,4,5,6,2)
```

también se puede crear una tupla que contenga solo un elemento, para esto se debe añadir una “,” después del elemento para que sea reconocido como una tupla.

```
Tupla_2 = (9,)
```

Una lista puede ser convertida en tupla o viceversa con el siguiente comando:

```
Lista = list(mi_tupla)
```

De esta forma se ha creado una lista llamada lista_1 en base a una tupla.

Si se quiere convertir una lista en tupla se debe hacer de la siguiente forma:

```
Tupla_3 = tuple(Lista)
```

La utilización de tuplas es necesario cuando sea crea un set de datos a los cuales no se les quiere modificar nada, es decir, solo sirven para lectura.

Set: también denominados conjuntos se utilizan principalmente en operaciones lógicas de la programación, como la unión de conjuntos, operación que se lleva a cabo con el operador “|”. También se puede realizar la operación intersección de conjuntos con el operador “%”.

Para crear un conjunto este se debe declara con llaves de la siguiente forma:

```
Conjunto_a = {1,2,3,4,5}
```

```
conjunto = {6,7,8,9,10}
```

Al realizar la operación unión de conjuntos entregara el siguiente resultado:

```
a|b
```

```
{1,2,3,4,5,6,7,8,9,10}
```

Los conjuntos tienen bastantes funciones que se pueden hacer con ellas, queda como sugerencia que a aquel al que le interese aprender mas sobre tuplas consulte documentos online que se encuentran disponibles en la red.

Diccionarios: Son un objeto para guardar información de mucho interés actualmente. Dentro de los diccionarios se pueden guardar listas e incluso otros diccionarios.

Este tipo de estructura se caracteriza por que para cada elemento, existe una clave asociada al elemento. Los elementos almacenados no necesariamente están ordenados.

Para declarar un diccionario este como los conjuntos se declara con llaves ({}), pero la forma de agregar la información es distinta. Los valores se van agregando en pares, en donde el primer elemento corresponde a la clave y el segundo al valor, por ejemplo:

```
Diccionario = {"pingüino": "antártica", "león": "África", "kiwi": "nueva Zelanda"}
```

Las claves (o llaves) del diccionario son pingüino, león, kiwi y los valores son antártica, África, nueva Zelanda. Esto se pueden obtener con el método:

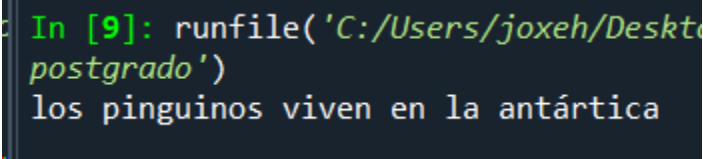
`Print (Diccionario.keys())` para las claves y

`Print (Diccionario.values())` para los valores

Los diccionarios se pueden imprimir (mostrar su contenido por pantalla) o también se puede imprimir un elemento en particular usando las claves del diccionario, por ejemplo utilizando un poco de lo ya aprendido, se puede imprimir el siguiente mensaje:

`Print ("los pingüinos viven en la "+ diccionario["pingüino"])`

Entregando el siguiente mensaje por pantalla (imagen 21):

A screenshot of a Jupyter Notebook cell. The prompt 'In [9]:' is followed by the code `runfile('C:/Users/joxeh/Desktop/postgrado')`. Below the code, the output is displayed as `los pinguinos viven en la antártica`.

```
In [9]: runfile('C:/Users/joxeh/Desktop/postgrado')
los pinguinos viven en la antártica
```

Imagen 21

Los diccionarios son estructuras mutables, por lo tanto, se pueden agregar o eliminar elementos. Para añadir elementos se debe hacer de la siguiente manera:

`Diccionario["panda"] = "corea"`

De esta forma se están agregando un elemento al final del diccionario.

Se pueden modificar elementos del diccionario en el caso de que sea necesario (se sobrescribe) por ejemplo.

`Diccionario["panda"] = "china"`

Eliminar elementos: Para eliminar elementos se debe especificar la clave a eliminar, de esta forma se elimina el par de información, esto se logra con el comando:

`del diccionario ["leon"]`

lo que entrega como resultado:

`{"pingüino": "antártica", "kiwi": "nueva Zelanda", "panda": "china"}`

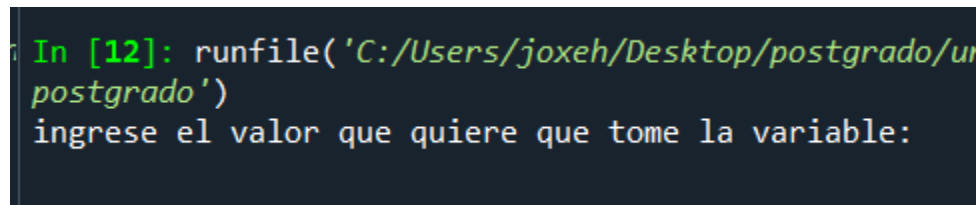
En un diccionario una clave puede tener mas de un valor por ejemplo:

Diccionario: {"pingüino": "antártica", "kiwi": "nueva Zelanda", "Australia", "panda": "china"}

Recibir datos por parte del usuario

Recibir datos por parte de usuario es fundamental a la hora de desarrollar programas, ya que estos datos serán ocupados en su funcionamiento. Estos datos recibidos son almacenados en variables para su posterior utilización. Para recibir datos se ocupa el método “input”, con el cual el programa espera recibir algo por parte del usuario. Por defecto, input transforma el dato ingresado en un String, por ejemplo:

```
Variable_1 = input ("ingrese el valor que quiere que tome la variable:")
```

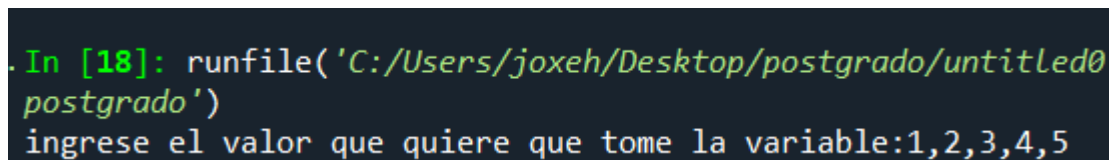


```
In [12]: runfile('C:/Users/joxeh/Desktop/postgrado/untitled0
postgrado')
ingrese el valor que quiere que tome la variable:
```

Imagen 22

El programa quedara en espera hasta que el usuario ingrese un valor o una cadena de texto.

Supongamos que los datos ingresados son los siguientes (imagen 23):



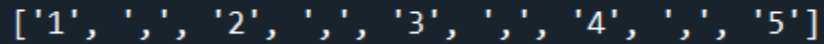
```
In [18]: runfile('C:/Users/joxeh/Desktop/postgrado/untitled0
postgrado')
ingrese el valor que quiere que tome la variable:1,2,3,4,5
```

Imagen 23

Como no se especifico el tipo de dato que se quiere recibir, Python asume que los datos son de carácter String, por lo que esas comas entre medio de los números **también son elementos**, por lo que al consultar el tamaño de la dimensión de “variable_1”, este devolverá el valor de 9. Aplicando un poco de lo aprendido podemos convertir esta variable llamada “variable_1” en una lista de la siguiente forma:

```
Variable_1 = list (Variable_1)
```

Al imprimir ahora variable_1 arroja lo siguiente por pantalla (imagen 24):



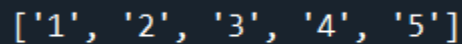
```
['1', ',', '2', ',', '3', ',', '4', ',', '5']
```

Imagen 24

En donde las comas entre comillas indican que son elementos de la lista los cuales están separados por coma uno de otro.

Si se quisiera quedar solo con los elementos “numéricos” de esta lista se puede hacer lo siguiente:

```
for elemento in Variable_1:
    if elemento == ",":
        Variable_1.remove(",")
    print(Variable_1)
    print()
print (len (Variable_1))
```



```
['1', '2', '3', '4', '5']
```

Imagen 25

Como se aprecia los caracteres “,” fueron eliminados quedando solo los caracteres “numéricos”. Esta parte se pensó de la siguiente manera, para todos los elementos que están contenidos en la variable_1 que sean iguales a “,” deben ser removidos. De esta forma el tamaño de la variable_1 disminuyo (imagen 25).

Los datos obtenidos son tipo String, pero estos pueden pasar a datos numéricos ocupando nuevamente un ciclo for de la siguiente forma (imagen 26):


```
@author: joxeh
"""
lista = ["1","2","3","4","5"]
lista2 = []
print(lista)
for elemento in lista:
    elemento = float(elemento)
    lista2.append(elemento)
print(lista2)
```

Imagen 26

De la lista obtenida anteriormente se convierten los valores String a numéricos siguiendo esta lógica: para cada elemento contenido en la lista, este se convierte en un valor numérico de tipo “float”, el cual es agregado a la lista vacía “lista2” previamente definida. Finalmente se imprime la lista2 entregando el siguiente resultado (imagen 27).

Al ejecutar esto Python arrojará lo siguiente:

```
[1.0, 2.0, 3.0, 4.0, 5.0]
```

Imagen 27

Datos que son de carácter numérico de tipo float.

Programa 1

Con todo lo aprendido a continuación se implementará un algoritmo que sea capaz de encontrar el Angulo faltante de un triángulo.

Primero debemos tener en consideración un poco de teoría básica de triángulos y ser capaces de recordar que en un triangulo la suma de los ángulos interiores es igual a 180° , además, si queremos calcular el Angulo que falta, es por que ya sabemos de ante mano el valor de los otros dos ángulos. Supongamos que los ángulos son "a", "b" son conocidos y "c" es el valor por encontrar, se tiene que:

$$a + b + c = 180^\circ$$

$$c = 180^\circ - a - b$$

Ahora, los ángulos a y b deben ser entregados, pero para eso le pediremos al usuario que ingrese los valores ya conocidos mediante la función input vista anteriormente. Con todo esto ya es posible crear un algoritmo capaz de encontrar el valor faltante. A diferencia de los otros aquí definiremos una función.

Para definir una función se debe ocupar la palabra reservada "def" seguido del nombre de la función y unos paréntesis, en donde dentro de estos paréntesis están definidas las variables que puede ocupar el programa, para este caso el programa no necesita los valores de otras variables ya que los valores de las variables serán pedidas por el propio programa. También esta función debe ser invocada (llamada) para que el programa se ejecute, si se omite la invocación el programa no correrá (no funciona). Llevando esto a la lógica de la programación queda (imagen 28):

```
@author: joxeh
"""
print("Bienvenido al programa para calcular el angulo desconocido")

def angulo():#Nombre y definicion de la funcion
    a =float(input("Ingrese el valor del primer angulo:"))
    b =float(input("Ingrese el valor del segundo angulo:"))
    c = 180-a-b
    print("El valor del angulo desconocido es:", c)

angulo() #Se llama a la funcion
```

Imagen 28

Se ha creado un programa capaz de recibir dos datos de entrada que corresponden a los valores de los ángulos y así calcular el ángulo faltante, queda como tarea poner condicione

de entrada a los valores para así no ingresar valores negativos y que la suma de los dos primeros ángulos sea menor a 180° .

Un algoritmo puede contener varios subprogramas que al unisonó funcionan como un todo. Desde ahora en adelante al crear un programa se definirá su nombre y su invocación.

Programa 2

A continuación se crear un entretenido programa que sea capaz de indicar cuanto te ama tu mascota. Para esto necesitamos números de entre [0-100] y el nombre de tu mascota.

Los números de los intervalos serán seleccionados al azar y si el numero corresponde a ciertos intervalos entregara un mensaje según tu nivel. Supongamos que creamos 3 escalas, la primera de [0-33[la segunda de [33 – 66[y la tercera de [66-100[. Para la primera escala mostrara el mensaje “considera dar mejores cuidados a tu mascota”, el segundo intercalo mostrara “podrías hacerlo mejor” y el tercer intervalo mostrara “buen trabajo! Tu mascota esta super!”

Ahora se necesita un numero aleatorio que valla de 0 a 100, para esto importaremos la biblioteca “random” biblioteca que se encarga de generar números aleatorios entre dos valores dados, en nuestro caso de 0 a 100.

En un principio habrá un condicional en donde se pregunte al usuario si quiere seguir adelante o no con el programa mediante opciones. Si el usuario elije no seguir el programa se cerrará mostrando un mensaje. En caso de seguir se generará un valor aleatorio que será evaluado en 3 condicionales diferentes.

Los condicionales que vayan evaluando los 3 casos posibles de la siguiente manera: Si el valor generado cumple con las condiciones del primer intervalo, entonces se ejecutara **esa condición** y no pasara a verificar la segunda la condición.

Si la primera condición no se cumple ira a la segunda a ver si coincide y se detendrá allí por el “elif” presente. Este elif hace que se excluyan los condicionales posteriores si en vez de un elif se coloca un “if” **ambas condiciones es de ejecutarán**, una después de otra por supuesto. Finalmente el condicional “else” indica que si no el valor generado no entra ni en la primera ni segunda condición se ejecute esa línea de código perteneciente al “else”.

Llevado lo explicado con anterioridad llevado a lenguaje Python queda (imagen 29):

```

5  @author: joxeh
6
7
8  """
9  import random
10 def felicidad():#Se define la funcion
11     print("Quieres saber cuanto te ama tu mascota?")
12     print("1) Si")
13     print("2) No")
14     respuesta = int(input("ingresa tu respuesta:"))#Espera la respuesta
15     print()
16     if respuesta ==1:
17         valor_generado = random.randint(0,100)
18         if 0 <= valor_generado <33:
19             print("Tu mascota te ama un",valor_generado,"%")
20             print("Considera dar mejores cuidados a tu mascota :(")
21         elif 34 <= valor_generado <66:
22             print("Tu mascota te ama un",valor_generado,"%")
23             print("Podrías hacerlo mejor!! ._.")
24         else:
25             print("Tu mascota te ama un",valor_generado,"%")
26             print("Buen trabajo! Tu mascota esta super! :D")
27
28
29     else:
30         print("Te dio miedo saber cuanto te ama tu mascota, verdad?")
31
32
33 felicidad()# Se invoca o llama la funcion
34

```

Imagen 29

Programa 3

También se puede generar algoritmo capaz de sumar numero una cantidad infinita de veces, para esto se ocupará la **recursividad de la función definida**. Una recursividad consiste en que una función puede llamar a otra o así misma si es requerida, a continuación se presenta el siguiente código, siguiendo básicamente la misma lógica del programa anterior se tiene que (imagen 30):

```

1  def primeros_sumandos ():
2      Primer_sumando = float(input("agregue el primer numero a sumar:"))
3      segundo_sumando = float(input("agregue el segundo a sumar:"))
4      suma = Primer_sumando+segundo_sumando
5      return suma
6  def agrega_sumando(suma):
7      print("lo que lleva de la suma es",suma)
8      print("desea agregar otro numero: ")
9      print("1)si")
10     print("2)no")
11     opcion = float(input("ingrese su opcion: "))
12     if opcion ==1:
13         n_sumando = float(input("agregue el numero a sumar:"))
14         suma = suma+n_sumando
15
16         agrega_sumando(suma)
17     elif opcion == 2 :
18         print("gracias por utilizar el programa, su suma total es de :",suma)
19         return suma
20
21     else:
22         print("ingrese una opcion valida: ")
23         agrega_sumando(suma)
24
25 suma = primeros_sumandos()
26
27 agrega_sumando(suma)
28

```

Imagen 30

- 1) Se define el nombre del programa y los parámetros que utiliza, como el paréntesis esta vacío, esto indica que esa función no requiere parámetros para su funcionamiento.
- 2) Se declara la primera variable la cual es "primer_sumando" la cual esta encargada de recibir el primer valor de tipo float por parte del usuario.
- 3) Se declara la segunda variable la cual es "segundo_sumando" la cual está encargada de recibir el segundo valor de tipo float por parte del usuario.
- 4) Se declara la variable suma la cual corresponde a la suma de primer_sumando y segundo_sumando
- 5) **se retorna el valor de suma.** Retornar un valor indica que este valor queda libre para ser utilizado por otros programas, es decir, el resultado retornado queda a disposición de otros algoritmos para que lo puedan utilizar.
- 6) se define la segunda función la cual es agrega_sumando la cual requiere un argumento, y este argumento es la suma obtenida anteriormente la cual fue retornada.
- 7) Esta línea de código se encarga de imprimir lo que se lleva sumado hasta ese momento
- 8) Muestra un mensaje preguntando si se quiere añadir otro sumando.

- 9) Muestra la primera opción a digitar
- 10) Muestra la segunda opción a digitar
- 11) Se almacena la opción en la variable opción. Esta es de tipo flotante
- 12) Se valida si la condición es verdadera o falsa. En caso de ser verdadera se ejecutará toda la línea contenida en ese if.
- 13) Pide ingresar el tercer sumando el cual queda almacenado en el tercer_sumando.
- 14) Se actualiza la suma. El valor de la suma pasa a ser el valor de la suma anterior mas el valor ingresado por el usuario.
- 15) Espacio en blanco
- 16) Se llama nuevamente a la función (recursividad) con el argumento actualizado y se repite todo lo anterior en caso de la respuesta sea "si".
- 17) Evalúa si la opción del usuario es "no", es decir, el usuario ingreso la opción número 2.
- 18) Imprime solo el valor de la primera suma junto a un mensaje.
- 19) Se retorna la suma (la variable suma) en caso de que se requiera en la utilización de otro programa.
- 20) Es un espacio en blanco
- 21) Es el ultimo condicional y solo se ejecuta si el usuario ingresa una opción distinta a 1 o 2.
- 22) Se muestra un mensaje de advertencia
- 23) Se ejecuta nuevamente el programa hasta que se valide la opción.
- 24) Espacio en blanco
- 25) Se invoca (llama o se instancia) la función para que se ejecute. Al resultado de lo que se obtiene en aquella función se le llamo "suma" es decir, de toda la ejecución del primer programa se pueden retornar una o más variables. En caso de ser retornadas más de una variable estas serán retornadas como tuplas ordenadas según el orden de aparición. Para este caso de como solo se retorna una variable no es necesario especificarle índice de la variable que necesitamos.
- 26) Espacio en blanco
- 27) Se instancia el segundo programa el cual necesita de la variable suma para funcionar, pero la variable suma se obtiene de la ejecución del primer programa (imagen 31).

```
agregue el primer numero a sumar:23
agregue el segundo a sumar:56
lo que lleva de la suma es 79.0
desea agregar otro numero:
1)si
2)no
ingrese su opcion: 1
agregue el numero a sumar:56
lo que lleva de la suma es 135.0
desea agregar otro numero:
1)si
2)no
ingrese su opcion: 2
gracias por utilizar el programa, su suma total es de : 135.0
```

Imagen 31

Resultado de la ejecución del código explicado anteriormente con la opción 1 ingresada por el usuario.

Lectura y escritura de archivos

Python nos permite leer y crear distintos tipos de archivos desde archivos de texto plano como de formato .txt hasta formatos xlsx (Excel).

Los archivos de formato .txt son los formatos más simples para almacenar información un ejemplo de ellos es el block de notas.

Existen 3 métodos fundamentales para manejar archivos:

W : Es el modo escritura del archivo (write). Permite escribir en un archivo lo que se requiera. Se le debe pasar el nombre del archivo, en caso de no existir el documento previo a la ejecución del programa se creará un nuevo documento en el directorio de ejecución del programa (imagen 32).

R: Es el modo de lectura del archivo, solo permite la lectura de este sin ninguna modificación. Si el nombre del archivo a leer no es encontrado, Python arrojara un error indicando que el archivo no se encuentra en la ruta de defecto (imagen 33).

Close: Cierra el archivo para seguridad de que no se pierda el contenido.

La siguiente imagen muestra un programa que escribe " lean el manual", es decir, ese es su contenido. Se pueden agregar mas frases o palabras. Si se quiere escribir en la línea que sigue (crear un salto de línea) solo se debe especificar el símbolo "\n" al principio de los caracteres que queremos que estén en la segunda línea. Si no se especifica el salto de línea, Python escribirá todo junto en una sola línea.

```

8
9  def creacion():
10     archivo = open ("manual.txt","w")
11     archivo.write("Lean el manual ")
12     archivo.close()
13  creacion()

```

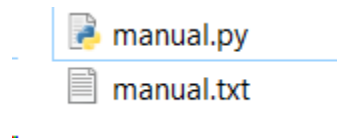


Imagen 32

El archivo manual.txt es creado en el directorio en donde se ejecutó el programa.

También se pueden leer archivos de la siguiente manera:

```

8  def creacion():
9     archivo = open ("animales.txt","r")
10    lineas = archivo.readlines()
11    print(lineas)
12    archivo.close()
13
14
15  creacion()
16

```

Imagen 33

8) se crea la función

9)se crea asigna una variable a la lectura del archivo, el cual leerá el archivo “animales.txt”. Es importante especificar la extensión del archivo después del nombre.

10)el archivo lee las líneas contenidas en el documento y esas líneas son asignadas a una nueva variable llamada “lineas”.

11) Imprime las líneas a medida que va leyendo el archivo.

12) se cierra el documento luego de la lectura.

Luego de la ejecución del archivo se ve esto por la pantalla imagen 34:

```

In [30]: runfile('C:/Users/joxeh/Desktop/postgrado/Defensa avance
Users/joxeh/Desktop/postgrado/Defensa avance
['perro\n', 'gato\n', 'leon\n', 'tortuga']

```


Imagen 34

En donde el elemento “\n” indica un salto de línea. Quiere decir que en el archivo animales los caracteres escritos en el están escritos uno abajo del otro pero no son parte del contenido (imagen 35).

El archivo también puede ser manipulado mediante el método “with” de la siguiente forma:

```
"""
def creacion():
    with open ("animales.txt","r") as archivo:
        contenido = archivo.read()
        lineas = contenido.split("\n")
        print(lineas)
creacion()
```

imagen 35

Obteniéndose la lista sin los \n:

```
Users/joxeh/Desktop/postgrado/Defensa
['perro', 'gato', 'leon', 'tortuga']
```

Imagen 36

En la documentación oficial de Python se encuentran todos los distintos métodos que se pueden aplicar en la manipulación de archivos.

Data frames

Un data Frame no más que la unión de distintos diccionarios los cuales representan diferentes tipos de información. Para cada columna existe una única clave, pero estas claves tienen distintos valores agregados **en forma de lista**. Por ejemplo supongamos los siguientes diccionarios (imagen 37):

```
2 def datos():
3     diccionario_1 = {"Nombres": ["Ricardo", "Sebastian", "Gabriel", "Michelle", "Eduardo"]}
4     diccionario_2 = {"Apellidos": ["Lagos", "Piñera", "Boric", "Bachelet", "Frei"]}
5     diccionario_3 = {"Años totales de gobierno": [6, 8, 4, 8, 6]}
6     datos()
```

Imagen 37

En donde cada diccionario tiene su clave correspondiente (“nombres”, “Apellidos” y “Años totales de gobierno”) y su vez cada clave tiene distintos valores asociados en forma de lista (imagen 38).

Ahora, se podría mostrar esta información de forma mas ordenada, como en una tabla de Excel por ejemplo. Para lograr esto se deben concatenar los diccionarios creando así un data Frame.

Para realizar esto se debe importar la biblioteca “pandas” la cual nos ayudara en la manipulación de información. Esta biblioteca de encarga de trasponer la información de modo que la llave quede como cabecera de la columna y los valores asociadas a la llave quedan como filas.

Modificando un poco el programa queda:

```
def datos():
    diccionario_1 = {"Nombres" :["Ricardo"," Sebastian","Gabriel"," Michelle","Eduardo"]}
    diccionario_2 = {"Apellidos": ["Lagos","Piñera","Boric","Bachelet"," Frei"]}
    diccionario_3 ={"Años totales de gobierno":[6,8, 4,8,6]}
    diccionario_4 =diccionario_1 |diccionario_2|diccionario_3
    frame = pd.DataFrame(diccionario_4)
    frame.fillna(0)
    print(frame)

datos()
```

Imagen 38

- 1) En esta línea se importa la biblioteca panda y esta será llamada como “pd”, es decir, Python al leer pd en el código entenderá que se trata de la función pandas.
- 2) Se define la función llamada datos la cual no requiere argumentos.
- 3) Diccionario que contiene una clave (Nombres) con diferentes valores
- 4) Diccionario que contiene una clave (Apellidos) con diferentes valores
- 5) Diccionario que contiene una clave (Años de gobierno) con diferentes valores
- 6) Unión de los 3 diferentes diccionarios generan un nuevo diccionario llamado diccionario_4. Esto se logra con el operador lógico de unión de conjuntos (|)
- 7) Se crea un data Frame llamado “frame” a partir del diccionario_4.
- 8) Se imprime el data Frame por pantalla para visualizar su información.
- 9) línea vacía
- 10) Se instancia al programa para que funcione.

Lo que se ve luego en pantalla es lo siguiente (imagen 39):

	Nombres	Apellidos	Años totales de gobierno
0	Ricardo	Lagos	6
1	Sebastian	Piñera	8
2	Gabriel	Boric	4
3	Michelle	Bachelet	8
4	Eduardo	Frei	6

Imagen 39

Por lo general un data Frame tiene de miles a cientos de miles de datos, por lo que realizar un análisis exploratorio siempre es una buena opción. Se puede imprimir la cabecera de los datos mostrando por pantalla los 5 primeros elementos del dataframe mediante el método “. head ()”, para esto se debe realizar el siguiente método:

`Nombre_dataframe.head()`

también podemos visualizar las características del conjunto de datos de tal manera que sepamos la cantidad de datos no nulos contenidos y que tipo de datos tiene el data Frame. Su aplicamos este método al dataframe creado anteriormente se tendría (imagen 40):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Nombres                               5 non-null      object
1   Apellidos                             5 non-null      object
2   Años totales de gobierno              5 non-null      int64
dtypes: int64(1), object(2)
memory usage: 248.0+ bytes
None
```

Imagen 40

Como se aprecia en la imagen se entrega el nombre de las columnas y el tipo de dato contenido por las columnas. Dype “object” quiere decir que son del tipo String, mientras que “int64 indica que son variables numéricas de tipo entero”.

Con lo data Frame se pueden realizar muchas operaciones en función de los datos que se tengan, también se pueden hacer graficas y un pretratamiento antes trabajar con ellos.

Data Frame con listas

Como se vio anteriormente, se pueden crear data Frame a partir de un diccionario, pero también se pueden crear data Frame a partir de listas. Supongamos que se tienen las siguientes listas (imagen 41):

```
1 import pandas as pd
2 def datos():
3     lista1 = [4,5,6,7,5.5,4.3]
4     lista2 = ["Miguel","Bruno", "Ruben","Sigrid","Felipe","Jaime"]
5     frame = pd.DataFrame(list(zip(lista1,lista2)), columns = ['Nota','Alumno'])
6     print(frame)
7
8
9 datos()
```

Imagen 41

- 1) Se importa la biblioteca pandas como pd
- 2) Se define la función datos la cual no necesita ningún argumento
- 3) Se define una lista la cual solo contiene elementos numéricos
- 4) Se define una lista la cual contiene nombres
- 5) Se crea un data Frame. Para esto se unen las dos listas como tuplas, es decir, a cada elemento de la lista1 se le asigna un elemento de la lista2, en donde ambas listas deben tener la misma longitud. Luego definir las listas como tuplas se le escribe el nombre que tendrá cada columna como encabezado siguiendo el orden de las listas ingresadas en la función zip. Los valores de la lista1 tendrán como encabezado de columna "Nota" y los elementos de la lista2 tendrá como encabezado el String "Alumno". Luego toda esta tupla generada se convierte en una lista y finalmente se transforma en dataframe mediante el método de pandas pd.DataFrame().
- 6) Se imprime el data Frame
- 7) línea vacía
- 8) línea vacía
- 9) Se instancia la función.

Finalmente se ve lo siguiente por pantalla(imagen 42):

	Nota	Alumno
0	4.0	Miguel
1	5.0	Bruno
2	6.0	Ruben
3	7.0	Sigrid
4	5.5	Felipe
5	4.3	Jaime

Imagen 42

también se puede mostrar una grafico de estos estos datos de la siguiente manera (imagen 43):

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 def datos():
4     lista1 = [4,5,6,7,5.5,4.3]
5     lista2 = ["Miguel","Bruno", "Ruben","Sigrid"," Felipe"," Jaime"]
6     frame = pd.DataFrame(list(zip(lista1,lista2)), columns = ['Nota','Alumno'])
7     print(frame)
8     print()
9     frame.plot(kind="bar", x="Alumno", y = "Nota")
10    plt.show()
11    datos()
```

Imagen 43

- 1) Se importa la biblioteca panda
- 2) Se importa la biblioteca matplotlib.pyplot como plt
- 3) Se crea la función
- 4) De define la lista
- 5) Se define la lista
- 6) Se crea el data Frame
- 7) Se imprime la información por pantalla
- 8) Se imprime un espacio vacío para que la información no se vea tan junta
- 9) Se usa el módulo .plot propio de la biblioteca matplotlib para graficar. En kind se selecciona el tipo de grafico los cuales puede ser de tipo: line, bar, barh, hist, box, kde, pie etc. también se deben definir cuales van a ser las variables x e y del gráfico. En caso de que el data Frame tenga más de dos columnas se deben seleccionar las columnas de interés. Al grafico se le puede poner leyendas o personalizarlo según nuestras necesidades.

Matrices

Las matrices son arreglos en dos dimensiones, se les puede considerar como una lista de listas. Al igual que las listas y tuplas son iterables, es decir, se pueden recorrer mediante el uso de ciclos.

Hoy en día existen varias maneras de crear matrices, como la manera clásica o las otras que son generadas mediante el uso de bibliotecas como la biblioteca numpy de Python.

A continuación se muestra la forma clásica (imagen 44):

```

1  import numpy as np
2  matriz_1 = [[1,2,3],
3              [4,5,6],
4              [7,8,9]]
5  for elemento in matriz_1:
6      print(elemento)

```

Imagen 44

Como se aprecia, la estructura coincide con la definición, ya que una matriz no es más que una lista de listas. Las matrices se pueden recorrer de varias formas diferentes mediante el uso del ciclo for. La matriz de la imagen superior es recorrida elemento por elemento, es decir, imprimirá por pantalla las listas que componen la matriz obteniéndose lo siguiente (imagen 45):

```

[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

```

Imagen 45

Se puede observar que se imprimieron las 3 listas según el orden de aparición.

Ahora si se quiere imprimir uno a uno los elementos de las listas que componen a la matriz hay que hacer uso de un **ciclo for anidado**. Un ciclo for anidado no es más que un ciclo for seguido de otro ciclo for, en donde por definición el primer ciclo for **asociado a las filas y el segundo ciclo a los elementos que componen las filas o sea las columnas**. Esto se aprecia en el siguiente ejemplo (imagen 46):

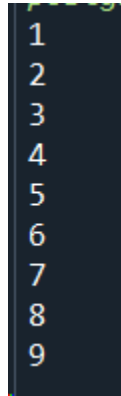
```

1  import numpy as np
2  matriz_1 = [[1,2,3],
3              [4,5,6],
4              [7,8,9]]
5  for fila in matriz_1:
6      for columna in fila:
7          print(columna)

```

Imagen 46

Al imprimir los elementos contenidos en cada fila se obtiene lo siguiente (imagen 47):



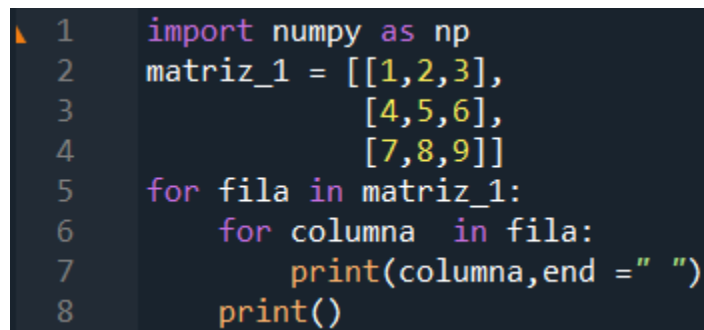
1
2
3
4
5
6
7
8
9

Imagen 47

Lo que tiene sentido ya que el primer ciclo for se ejecutara hasta que termine con todos los elementos de la primera fila. La primera posición de una matriz es la posición [0][0], posición en la que se encuentra el número 1, la posición [0][1] esta ocupada por el número 2 y así sucesivamente hasta el número 9 el cual está ocupando la posición [2][2].

Primeramente el ciclo for se detiene en la primera fila, luego el segundo ciclo for anidado va recorriendo los elementos de esa fila, una vez que termine de ejecutarse el ciclo for anidado, el primer ciclo for pasa a la segunda fila y así sucesivamente. Cabe mencionar que mientras mas anidado este el ciclo for más veces se ejecutara ese ciclo.

también se puede imprimir lo anterior de forma de una matriz de la siguiente manera:



```
1 import numpy as np
2 matriz_1 = [[1,2,3],
3             [4,5,6],
4             [7,8,9]]
5 for fila in matriz_1:
6     for columna in fila:
7         print(columna,end = " ")
8     print()
```

Imagen 48

La diferencia con el anterior es que se añadió el parámetro "end" (imagen 48) el cual es utilizado para imprimir un carácter luego de lo que se desea imprimir, y como el " " (espacio en blanco) es un carácter, este aparecerá después de los números. Luego de imprimir los elementos de la primera fila se imprimirá un espacio en blanco, esto se logra con el uso del Print (), pero sin argumento (si se omite el Print del final, todos los elementos serán impresos como una sola fila). Una vez que se imprime el espacio en blanco como salto de línea, se imprimen todos los valores de la siguiente fila y así sucesivamente hasta que se terminen de ejecutar los ciclos for (imagen 49).

1	2	3
4	5	6
7	8	9

Imagen 49

también se puede modificar el programa de forma que sume los elementos de una matriz siempre y cuando estos sean de carácter numéricos, de la siguiente forma (imagen 50):

```
1  import numpy as np
2  matriz_1 = [[1,2,3],
3              [4,5,6],
4              [7,8,9]]
5  acumulador = 0
6  for fila in matriz_1:
7      for columna in fila:
8          print(columna,end = " ")
9          acumulador = acumulador+columna
10     print()
11     print()
12     print(acumulador)
```

Imagen 50

En la línea:

5) se define una variable “acumulador” la cual está definida con el valor 0

9) el acumulador incrementa de valor sumando los elementos (columna) al acumulador. El acumulador vale 0 por defecto pero se le suma el primer elemento, es decir, $0+1$ es 1, por lo tanto el acumulador ahora cambio de valer 0 a valer 1, luego al acumulador se le suma el siguiente elemento, el 3 haciendo la operación de $1+2 = 3$, cambiando el valor de 1 a 3 y así sucesivamente hasta alcanzar el valor de 45, valor que corresponde a la suma de todos los elementos de la matriz.

Crear matrices a partir de numpy

La ventaja de crear matrices con numpy es que nos ahorra gran cantidad de código a la hora de programar y también facilita también las operaciones.

Volamos al ejemplo anterior (imagen 51):


```

1  import numpy as np
2  matriz_1 = [[1,2,3],
3              [4,5,6],
4              [7,8,9]]
5  for fila in matriz_1:
6      for columna in fila:
7          print(columna,end = " ")
8      print()

```

Imagen 51

Una forma más fácil de definir esta matriz usando numpy (imagen 52):

```

import numpy as np
matriz_1 = [[1,2,3],
            [4,5,6],
            [7,8,9]]

matriz = np.array(matriz_1)

```

Imagen 52

Se definió una matriz llamada “matriz” en numpy, ahora esta matriz creada puede ser usada para distintas operaciones. Supongamos que queremos sacar su traspuesta. Para esto usamos uno de los métodos incorporados en numpy, el método. `transpose()`. Por lo general cuando se ocupa un método, este va después de un punto. La definición algebraica de matrices no dice que cuando se traspone una matriz, las filas pasan a ser columnas y viceversa.

Al aplicar la operación transpose se obtiene lo siguiente (imagen 53):

```

[[1 4 7]
 [2 5 8]
 [3 6 9]]

```

Imagen 53

La matriz se traspuso sin mayores complicaciones gracias a los métodos incorporados para trabajar con matrices de numpy.

también se puede sumar y multiplicar matrices que sean objetos numpy, siempre y cuando se respeten las dimensiones que establece la matemática para la operación de matrices.

Además con numpy se pueden crear distintas matrices de interés como por ejemplo, la matriz identidad, una matriz llena de cero o de 1.

Matriz identidad: Es la matriz que solo contiene “1” en su diagonal principal. Para crearla solo basta indicarle la dimensión que queremos que sea la matriz junto al “método.identity()”. Llevado esto a la programación se ve así (imagen 54):

```
1 import numpy as np
2 def matriz():
3     matriz_identidad = np.identity(8)
4     print(matriz_identidad)
5
6 matriz()
```

Imagen 54

Y en su salida se tiene (imagen 55):

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

Imagen 55

Matriz de 0 y 1: Sigue la misma lógica anterior pero cambian los métodos con que se hacen (imagen 56).

```
1 import numpy as np
2 def matriz():
3     matriz_ceros = np.zeros((8,10))
4     print(matriz_ceros)
5
6 matriz()
```

Imagen 56

Sintaxis para la creación de una matriz llena de ceros.

- 1) Se importa la biblioteca numpy como np

- 2) Se define la función llamada “matriz”
- 3) Se asigna una variable al método encargado de crear la matriz. Hay un doble paréntesis para insertar las dimensiones de la matriz. El primer argumento corresponde **al número de filas y el segundo al número de columnas**.
- 4) Se imprime la matriz de ceros
- 5) Espacio en blanco
- 6) Se instancia al programa para que inicie.

Visualmente se obtiene lo siguiente (imagen 57):

```
[[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. 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.]
 [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. 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.]
```

Imagen 57

Para obtener una matriz llena de “1” solo se debe cambiar el método. En vez de colocar “.zeros”, se debe colocar “.ones”

Un método interesante para el algebra de matrices es calcular su determinante, para esto se utiliza el método “.linalg.det()”, en donde dentro del paréntesis va la variable matriz imagen (57).

```
1
2  def matriz():
3      import numpy as np
4      matriz_1 = [[-3,2,4],
5                  [-4,2,4],
6                  [1,-1,1]]
7
8      matriz = np.array(matriz_1)
9      print(matriz)
10     print()
11     determinante = np.linalg.det(matriz)
12     print("Su determinate es", determinante)
13  matriz()
```

Imagen 57

```

[[-3  2  4]
 [-4  2  4]
 [ 1 -1  1]]

Su determinate es 6.0

```

Imagen 58

Aplicación de las matrices en el proceso de simulación por Montecarlo

En el programa desarrollado para la determinación de vida útil ubo muchos cálculos por medio de matrices, y estas matrices eran de enormes dimensiones como por ejemplo, 5 millones de filas. A modo de explicación del código desarrollado, se comentará la línea de código que relaciona 2 matrices, una es la matriz que contiene la Aw (imagen 60), y otra matriz que contiene la cantidad de humedades asociadas a la monocapa (imagen 59). Para este ejemplo se hizo una simulación 5 veces con lo que se obtuvieron las siguientes matrices:

```

humedad = [[0.11335597 ,0.23864904, 0.32180365, 0.43077414 ,0.51133134 ,0.5580707,
             0.69123713, 0.75212001, 0.76643976, 0.8017674 ],
            [0.11481564, 0.23405021, 0.32272228, 0.43176221,0.510407,0.55713715,
             0.69204174, 0.75155164, 0.76299369, 0.80200762],
            [0.11202758, 0.24020198, 0.32111203, 0.43262172, 0.50989991, 0.55711468,
             0.69061617, 0.75341894, 0.76277486, 0.79976601],
            [0.11281773, 0.23890569, 0.32201632, 0.4306848, 0.50985959, 0.55811382,
             0.68877663, 0.75253598, 0.76429474, 0.8019184 ],
            [0.11309346, 0.23927626, 0.32337425, 0.43183616, 0.50940966, 0.55704745,
             0.69113649, 0.75171193, 0.76475097, 0.79953884]]

```

Imagen 59

```

21
22  aw = [[16.728549847655366, 17.445321490778706, 16.35452552580076, 16.175185648930217, 17.162293
23         24.15277796067443, 35.33847245689735, 20.494635487475936, 34.2599947811073, 35.639138449
24         0.7913158442452449, 0.7054160019564086, 0.7843454727165952, 0.9270501138476921, 0.750263
25         0.11335596533497146, 0.11481563755398443, 0.11202757903756635, 0.11281772936192515, 0.11
26         0.23864903910524574, 0.23405021419946007, 0.24020197593769615, 0.23890568879558466, 0.23
27         0.3218036490439465, 0.3227222805411478, 0.32111203368237096, 0.3220163197472117, 0.32337
28         0.4307741357059425, 0.431762211066511, 0.43262171987519976, 0.43068479846521374, 0.43183
29         0.5113313446861755, 0.5104070046385096, 0.5098999083945828, 0.5098595863240754, 0.509409
30         0.5580707014822727, 0.5571371481625806, 0.557114678270936, 0.5581138222175674, 0.5570474
31         0.6912371307114944, 0.6920417396804016, 0.6906161666494246, 0.6887766253981631, 0.691136
32         0.7521200061002477, 0.751551638533804, 0.7534189449714676, 0.752535979193633, 0.75171192
33         0.7664397645648932, 0.7629936945280325, 0.7627748620769753, 0.7642947362983077, 0.764750
34         0.801767398912695, 0.8020076216920055, 0.7997660109818296, 0.8019184021301787, 0.7995388

```

Imagen 60

La matriz A_w es más extensa pero en la foto no se visualiza totalmente. Esta matriz además tiene 3 filas extras, filas que corresponden a las constantes de GAB para la isoterma del tomate a 30 °C.

Los datos de A_w y humedad de monocapa deben ingresar en la ecuación de isoterma de Gab, para obtener el valor de las humedades, a su vez, estas humedades son coleccionadas en una lista a medida que se van generando.

$$M_e = \frac{M_o * C * K * A_w}{(1 - K * A_w) * (1 + (C - 1) * K * A_w)}$$

Ecuación de isoterma de GAB

Como se tiene todas las variables, estas se pueden reemplazar para así obtener la humedad de equilibrio. Esto llevado a lenguaje de programación se ve así (imagen 61):

```

36
37     matriz5 = []
38     l = 0
39     for i in range(len(humedad)):
40         l = l+1
41         for j in range (len(humedad[0])):
42             sal = humedad[i][j] ##### la transpuesta va si o si
43             mono = (aw[0][l-1]*aw[1][l-1]*aw[2][l-1]*sal)/((1-aw[2][l-1]*sal)*((1-aw[2][l-1]*sal+aw
44             matriz5.append(mono)
45             #print ("sal = ",sal,"mono =",matriz4[0][l-1],"ca = ",matriz4[1][l-1],"k: ",matriz4[2][l-1]
46             print("Aqui esta la matriz 5 la cual son las iteraciones con la ecuacion de Gab")
47             print()
48             print(matriz5)
49
50     return matriz5,humedad
51     triz()

```

```

*sal+aw[1][l-1]*aw[2][l-1]*sal)))

```

Imagen 61. La imagen de abajo corresponde al pedazo del código que no se encuentra en la imagen superior (línea 43)

37) se define una lista vacía para almacenar los valores que se vaya generando.

38)se define un contador igual a 0

39) se define un ciclo for para recorrer los elementos generados con la función range. Esta función crea una tupla de elementos desde 0 hasta n-1. Para este ejemplo la cantidad de elementos que tiene la matriz humedad es de 50. Por lo que len creara una tupla de elementos desde 0 hasta 49. Se hace esto con el fin que estos números se conviertan en uno de los índices de posición de la matriz.

40) el contador l se le suma 1 luego de que las condiciones de abajo del contador se cumplan para todos los elementos.

41) se utiliza otro ciclo for para que este recorra los elementos de los elementos los cuales son 10. El for anterior recorrido todos los elementos y por eso el valor es mas alto, en cambio, el valor del ciclo for de la línea 41 es mas pequeño debido a que **recorre la lista contenida en la lista**. Para este caso la matriz humedad tiene 50 elementos, pero estos 50 elementos están contenidos en listas de 10 elementos y ese es el valor que entrega este ciclo for. El [0] indica la lista que quiere ser contada, pero se puede utilizar cualquier índice del 0 al 9.

42) se declara la variable “sal” que en realidad es la aw asociada a la sal. El valor de sal va cambiando a medida que cambia de posición. Su primera ubicación sera la [0][0] luego las [0][1] y así sucesivamente hasta llegar a la dimensión [49][49].

43) La variable “mono” es la humedad de equilibrio calculada con la ecuación de isoterma de gab, por lo que al lado derecho está escrita la ecuación de GAB en lenguaje Python junto a las variables que deben ser reemplazadas. Las constates de la isoterma ocupan lugares específicos dentro de la matriz Aw, y estos lugares corresponden a las 3 primeras filas de la matriz.

$$sal = sal, mo = Aw[0][l - 1], c = Aw[1][l - 1], k = Aw[2][l - 1]$$

por lo tanto las filas se mantienen fijas, pero los elementos columna van cambiando gracias a la ayuda de un contador el cual cambia en 1 luego de recorrer todos los elementos de la matriz aw. Una vez que se recorran todos lo elementos la matriz humedad cambia de la posición 0 a la posición 1 y así sucesivamente hasta completar todas las iteraciones que fueron seleccionadas por el usuario.

44) Se guardan los elementos generados en una lista declarada previamente

45) Línea comentada. No es tomada en cuenta por el intérprete.

46) se imprime un mensaje

48) se imprime la matriz con los resultados

50) se retornan las variables de interés

51) se instancia la función.

Finalmente se obtiene una lista con las humedades de equilibrio de la isoterma de GAB para un numero de simulación igual a 5 (imagen 62).

```
[12.939997601477963, 17.509331829099427, 20.01793637890827, 23.498822547656513,
26.484031584674668, 28.4681912749729, 35.70304239543793, 40.18998413433035, 41.39963980221478,
44.69660325889049, 14.369052697790757, 18.27943723834444, 20.608887987822065, 23.5632586893974,
25.954971795569538, 27.537369836607173, 33.10259669813086, 36.22168802562664, 36.88361508530914,
39.319325924037216, 11.90171133971334, 16.65107127790241, 19.093052375567854, 22.606594852202356,
25.39553315338844, 27.329193497790826, 34.26900373583534, 38.67456780381961, 39.4196578644297,
42.644086602815555, 14.452838687722936, 18.843410628583676, 21.578797348789312,
25.792973287735766, 29.706110009026524, 32.6283966299132, 44.02102229496449, 52.827892366946266,
54.83857537632385, 62.41238663274399, 14.396650325423659, 18.539819762697427, 20.834960097701156,
23.983538963969654, 26.5739846091806, 28.376143044774814, 34.74082940583394, 38.525702387191714,
39.44267018162127, 42.103197703620275]
```

Imagen 62

Uso del programa

El programa de fue desarrollado para llevar a cabo una simulación probabilística de Montecarlo. Su uso es sencillo ya que el usuario solo debe ingresar datos desde el teclado.

Luego de que se tenga instalado Python y todas las bibliotecas necesarias para su funcionamiento, se hace correr el programa mediante el intérprete “Spyder”.

Al iniciar el programa se ve el siguiente mensaje (imagen 63):

```
Bienvenido al programa de simulacion mediante Montecarlo
introduce el numero de filas, estas corresponderan a las repeticiones de montecarlo) :
```

Imagen 63

En donde el usuario debe ingresar un numero mayor a 0 para comenzar a realizar la simulación. Se recomienda ingresar un valor mayor a 500. El programa trabajo hasta con 5 millones de simulaciones, tardando eso si alrededor de 6 horas en terminar su ejecución.

Luego de ingresar el número de simulaciones (escribiendo el número y presionando Enter), el programa pide el numero de variables a utilizar. La ecuación de Gab cuenta con 3 parámetros, por lo tanto, estas se den considerar en la sumatorio de variables, es decir, si se quiere trabajar con 8 variables, se deben agregar en total 11 variables, ya que $3+8 = 11$ (imagen 64).

```
introduce el numero de columnas: (variables del experimento, las 3 primeras corresponden a los
parametros de GAB, por ende, SI desea ingresar 4 sales, debera ingresar un total de 7 variables) : 10
```

Imagen 64

Posteriormente se pide ingresar el nombre (imagen 65) promedio y desviación de las variables de la siguiente forma.

```

ingrese el nombre de la variable
fila 1- columna 1 :Mo
ingrese el nombre de la variable
fila 1- columna 2 :C
ingrese el nombre de la variable
fila 1- columna 3 :k
ingrese el nombre de la variable
fila 1- columna 4 :LiCl
ingrese el nombre de la variable
fila 1- columna 5 :CH3COOK|

```

Imagen 65

Luego de esto pide ingresar los datos del promedio y de la desviación de las variables. Los datos no deben ser ingresados al azar debido a que los valores de las constantes están programados de forma tal que lo primero que se debe agregar son las constantes de Gab (M, C, K) en ese orden y luego la Aw asociadas a las sales en orden creciente. Es importante señalar que los números decimales deben ser escritos **con punto y no con coma**.

Luego de ingresar los datos, se muestra por pantalla una serie de valores en forma de par ordenado de la siguiente forma (imagen 66):

```

Esta matriz corresponde a las sales involucradas-Los valores de AW de cada sal - Y los valores
promedio de g de agua / g de solidos secos (m) (De izquierda a Derecha)
[['1' '' '0.11327775488347144' '0.13473036013411277']
 ['2' '' '0.23944891843352073' '0.1787562541934754']
 ['3' '' '0.3221398888815246' '0.20311906616083555']
 ['4' '' '0.43242972635139176' '0.2380506758280606']
 ['5' '' '0.5106419533050441' '0.2670610219250466']
 ['6' '' '0.5576866905220881' '0.28716890064389994']
 ['7' '' '0.6914570221525016' '0.3615157446472047']
 ['8' '' '0.7520915960798404' '0.40795089052096']
 ['9' '' '0.762547876035143' '0.4172479664254611']
 ['10' '' '0.8013211924510953' '0.4550794616966489']]

la ecuacion de la recta es: y = 0.45406385580468844 X + 0.05972480257491167

```

Imagen 66

Donde el usuario debe seleccionar los datos que el estime conveniente en caso de que se trabaje con otra matriz alimenticia. Como en el trabajo de Escobedo avellaneda utilizaron que la humedad inicial del tomate era 0.8 y su humedad de equilibrio de aproximadamente 0.65, se debe utilizar los elementos desde la posición 7 a la posición 10 (imagen 67). Estos elementos deben ser copiados y pegados como sigue a continuación:


```
A continuacion seleccione la actividad del agua(Aw) a utilizar,esta se encuentra en el eje X del grafico, asi como los datos del eje Y (g de agua/ g de solido seco)
Guiese por la grafica obtenida anteriormente
```

```
ingrese el valor de la coordenada X: 0.6914570221525016
True
ingrese el valor de la coordenada Y: 0.3615157446472047
```

Imagen 67

Luego de ingresar los datos, se deben ingresar los datos para ser reemplazado en la ecuación de vida útil y así obtener el primer valor determinista (imagen 68).

```
a continuacion, ingrese los datos termodinamicos
ingrese la Aw inicial:0.8

ingrese la perdida de humedad en %:10

ingrese la Aw de equilibrio:0.65
0.37267075176595776

a continuacion se calculara el valor de permencia, este es el wvptr/ gradiente de presion de vapor
para ello deberas agregar el valor del WVTR y luego la presion de vapor
ingrese el valor de wvtr en g/m2 *dia:1.222

ingree el valor de la gradiente de presion de vapor en mmhg:21.4020

ingrese el area del envase en m2:0.06

ingrese el valor del peso inicial em gramos100
```

Imagen 68

Luego de ingresar todos los datos solicitados se obtiene lo siguiente (imagen 69):

```
la duracion del producto es 242.81487460225097 dias, lo que es lo mismo que decir 8.093829153408366
meses de duracion

Este valor corresponde a un valor numerico determinista, por lo tanto, hay que aplicar test de
confianza para estimar un valor más real
A continuacion, se calculara mediante MONTECARLO iteraciones para la vida util del alimento mediante
las sales de interes asociadas a una actividad de agua Aw
ingrese la posicion desde la cual quiere empezar a trabajar, posicion desde la 1- 10
```

Imagen 69

Luego se da la opción de obtener más valores de vida útil reutilizando las generadas con Montecarlo:

```

vida util del alimento mediante las sales de interes asociadas a una
actividad de agua Aw
ingrese la posicion desde la cual quiere empezar a trabajar, posicion
desde la 1- 10

[['1' 'LiCl' '0.11300433509823361' '0.1379943657365662']
 ['2' 'CH3COOK' '0.23807908999779118' '0.1809667191050292']
 ['3' 'MgCl2' '0.3220286784913034' '0.20504645395250012']
 ['4' 'K2CO3' '0.4319529474771537' '0.2464796205928584']
 ['5' 'MG(NO3)2' '0.5099879891134335' '0.2764060569348775']
 ['6' 'NABR' '0.5579910474594328' '0.2987623358736695']
 ['7' 'SrCl2' '0.6910234691138165' '0.3824077922256604']
 ['8' 'NaCl' '0.7530749410419273' '0.4387518086429369']
 ['9' 'NH4CL' '0.7629300456096308' '0.44926007645545396']
 ['10' '(NH4)2SO4' '0.8010703871540183' '0.4950920945647046']]

ingrese su opcion desde la cual quiere trabajar:

```

Imagen 70

Se debe ingresar un número del 1 al 10. Como se trabajo anteriormente desde la posición 7 a la 10, se seleccionará el numero 7 ya que por defecto el programa toma todos los valores desde el 7 hasta el final, es decir, del 7 al 10 (imagen 70).

```

el maximo de dias calculados es 295.10114491751716 y el minimo de dias es 214.20776669851296
el maximo de meses calculados es 9.836704830583905 y el minimo de meses es 7.140258889950432

El promedio de dias de duracion es de: 243.9396900027806
Dias validos 100
Estos son los dias indeterminados: 0
Estos son los dias negativos: 0
Este es el porcentaje de dias sin usos 0.0
elija los intervalos de dias por los que quiere contar:

```

Imagen 71

Finalmente se entrega un breve reporte estadístico de los datos generados y el valor de vida útil calculado. también se da al usuario la opción elegir el intervalo de días que más le acomode. Em la literatura ocuparon un ancho de diez días (imagen 71).

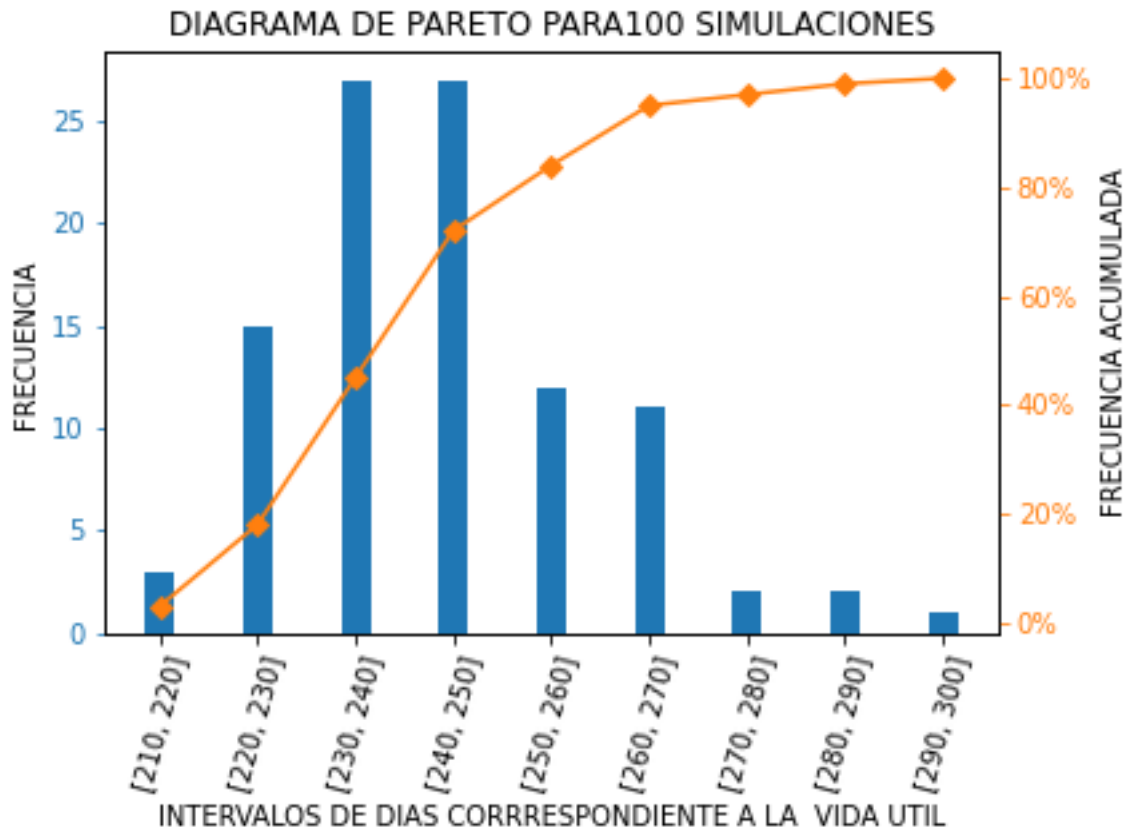


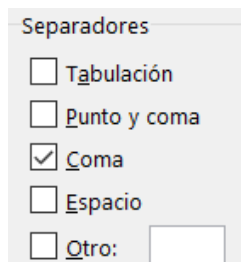
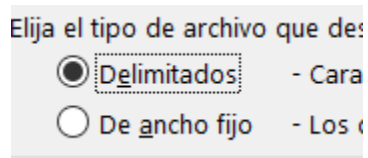
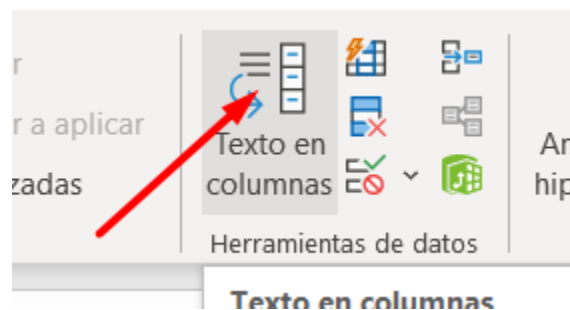
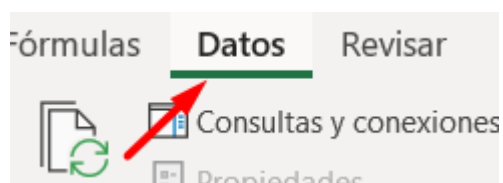
Imagen 72

Finalmente se obtiene un gráfico de frecuencias de ancho seleccionado (imagen 72) por el usuario junto a un archivo CSV en el mismo directorio desde donde está almacenado el programa.

Para manipular el archivo CSV se debe hacer lo siguiente:

- 1) Primero se deben seleccionar todos los datos con CTRL +SHIFT+flecha abajo del teclado.
- 2) Luego pinchar en “datos” e ir a “texto en columnas”.
- 3) Se selecciona delimitado y se da clic en siguiente
- 4) Se selecciona el separador por comas y se da clic siguiente ahora los datos quedan listos para futuras operaciones.

	A	B	C
1	,INTERVALO_DE_DIAS,FRECUENCIA,frecuer		
2	0,"[0, 10]",7,3.5e-06,3.5e-06		
3	1,"[10, 20]",6,3e-06,6.5000000000000004e-		
4	2,"[20, 30]",1,5.5e-06,1.2e-05		
5	3,"[30, 40]",7,3.5e-06,1.55e-05		
6	4,"[40, 50]",1,7.5e-06,2.3e-05		
7	5,"[50, 60]",5,2.5e-06,2.55e-05		
8	6,"[60, 70]",9,4.5e-06,3e-05		
9	7,"[70, 80]",1,8.5e-06,3.85e-05		
10	8,"[80, 90]",1,8e-06,4.65e-05		
11	9,"[90, 100]",2,6.1.3e-05,5.949999999999999		
12	10,"[100, 110]",2,6.1.3e-05,7.25e-05		
13	11,"[110, 120]",2,9.1.45e-05,8.7e-05		



Modelamiento de datos

A continuación explicara un modelo matemático desarrollado en Python para modelar procesos de deshidratación de alimentos. El modelo utilizado corresponde al de Midilli-Kucuk.

Para esto se creo un archivo CSV el cual contiene los datos de deshidratación respecto al tiempo. Las cabeceras de las columnas, es decir sus etiquetas, debe ser definidas por el usuario.

```
2 import os
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from sklearn.preprocessing import PolynomialFeatures
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 import csv
9 import numpy as np
10 import scipy as sp
11 from scipy.optimize import curve_fit
12 import math
13 from math import exp
14 from math import pi
15 import warnings
16 from sklearn.linear_model import LinearRegression
17 from scipy.optimize import curve_fit
18 from lmfit import Model, Parameters
19
20 def inicio():
21     datos = pd.read_csv("conda.csv")
22     print(datos.info()) ##Muestra el tipo de datos
23     print(datos.head())## Muestra los primeros 5 datos (encabezado)
24     print(datos.describe())##descripcion estadística de los datos
25
26     x= datos.tiempo.values.reshape(-1,1)
27     y = datos.mr.values.reshape(-1,1) ####Redimensiono al tiro los valores
28     #Para poder utilizarlos en los modelos
29     #x = np.linspace(0,450,40)#### Crea un arreglo donde los parametros son (punto de partic
30     #plt.title("MOISTURE RATIO VS TIME (WITH TIME OF MINUTES)")
31     #fig,axes = plt.subplots()
32     ##axes.scatter(datos["tiempo"],datos["hr"]) Entrega los datos como puntos
33     plt.figure(figsize=(9,5))###Se le da las dimensiones al gráfico, (ancho, alto)
34     #Face color cambia el color de la ventana
35
36     plt.plot(x,y,"r") #Cambia el color de la línea
37     plt.title("Humedad vs tiempo ")
38     plt.ylabel('Contenido de Humedad')
39     plt.xlabel('Tiempo (minutos)')
40     plt.legend(["contenido de humedad"],loc=3) #por defecto coloca la leyenda arriba
41     #Si no se colocan corchetes no se muestra el mensaje entero
```

Imagen 73

De la línea 2 a línea 18 se importan todas las bibliotecas que serán utilizadas (imagen 73), las bibliotecas que tienen el símbolo de precaución indican que esas bibliotecas no fueron utilizadas por el programa, por lo que pueden omitirse.

20) se declara la función llamada inicio, la cual no recibe argumentos

21) se lee un archivo CSV llamado "conda.csv" el cual es asignado a una variable llamada datos.

22)-24) se utilizan distintos métodos para obtener una descripción inicial de los datos, como los valores máximos, mínimo, la media, los tipos de datos y sí que existe algún elemento vacío dentro del archivo.

26 y 27) se le asignan las variables x e y según corresponda y se redimensionan los datos. Es importante señalar que la etiqueta de los datos debe coincidir con el nombre dado en los datos del archivo, sino arrojará error.

33) se declara el tamaño que tendrá la figura a mostrar, los números en paréntesis indican el ancho y el alto de la figura.

36) se grafican las variables antes asignadas. El r entre comillas indica el color de la línea del gráfico y se puede cambiar por el color azul ("b"), amarillo (y) etc.

37) se le coloca un título a la figura

38 y 39) se les pone el nombre a los ejes

40) Se le coloca leyenda a la figura. El numero después de la leyenda indica la posición en donde se quiere colocar la leyenda.

42) se muestra la figura por pantalla

44) se retornan las variables

```
47
48
49     def ecuacion_2(x,k,b,n):
50         return 0.9835*np.exp(-k*x**n)+b*x
51
```

Imagen 74

49) se define otra función llamada ecuación_2 (imagen 74). Si bien Midilli Kucuk contiene 4 variables, el primer termino corresponde al valor de humedad en el tiempo 0. Por lo tanto solo es necesario encontrar 3 variables.

50) Se retorna la definición de la ecuación para ser utilizado por otras funciones.

```

54 def midilli():
55     datos = pd.read_csv("conda.csv")
56     x= (datos.tiempo.values.reshape(-1,1))
57     y = datos.mr.values.reshape(-1,1)
58     #print(x)
59     #print(y)
60     model = Model(ecuacion_2,independent_vars=['x'], param_names=["k","b","n"])
61
62     params = Parameters()
63     params.add("k", value= 0.000001,min=0.000001,max = 2)
64     params.add("b", value=-0.0000001,min = -0.00001,max = -2)
65     params.add("n", value= 0.000001,min= 0.000001,max = 2)
66
67     result = model.fit(data=y, params=params, x=x)
68     #print(model)
69
70     print(result.fit_report())
71

```

Imagen 75

54) se define otra función llamada Midilli (imagen 75) y se repite el proceso ya explicado anteriormente.

60)se define la variable “model”, la cual ocupa la biblioteca model de “Lmfit”, biblioteca especializada en el ajuste de ecuaciones por medio del método de mínimo cuadrados. Model se le debe entregar la ecuación a ser modelada (ecuación que se definió anteriormente), se debe definir la variable independiente, para este caso es “x” y luego se le debe entregar los parámetros a ser encontrados, parámetros que están definidas como Strings y en una lista.

62) se declara una variable llamada “params” la cual hace uso del método Params() para así realizar una variación de parámetros de manera posterior.

63) a 65) se definen los valores iniciales propuestos por el usuario para encontrar los valores de las constantes. **Este paso es el más complicado de los programas de modelamiento por este método, ya que se esta utilizando el tanteo a la hora de encontrar los mejores valores, por lo que esto es la limitante a la hora de encontrar las constantes, sin embargo, una vez encontrados los valores el ajuste será excelente.**

67) Se declara una variable llamada “result” la cual es el producto del ajuste que realizo la biblioteca con los datos reales.

77) Esta línea de código muestra el reporte obtenido luego del ajuste. Este reporte nos ayudara a saber si vamos bien encaminados o no en la modelación, esto mediante al valor de R y el valor de Chi cuadrado, así como el porcentaje de error asociados a las constantes (imagen 76) .

El reporte se vería así:

```
[[Fit Statistics]]
# fitting method      = leastsq
# function evals      = 184
# data points         = 39
# variables            = 3
chi-square            = 3.1827e-18
reduced chi-square    = 8.8409e-20
Akaike info crit      = -1708.14203
Bayesian info crit    = -1703.15135
R-squared             = 1.00000000

[[Variables]]
k:  5.9613e-04 +/- 1.7507e-12 (0.00%) (init = 1e-06)
b: -1.6211e-05 +/- 1.4491e-13 (0.00%) (init = -1e-05)
n:  1.15710000 +/- 4.9824e-10 (0.00%) (init = 1e-06)

[[Correlations]] (unreported correlations are < 0.100)
C(k, n) = -0.994
C(b, n) = 0.749
C(k, b) = -0.684
```

Imagen 76

```
96 def comparacion_2(x_convertida,y):
97
98
99     k = 5.9613e-04
100
101     b = -1.6211e-05
102
103     n = 1.15710000
104
105
106
107
108     y_convertida3 = []
109     for elemento in x_convertida:
110
111         valor = ecuacion_2(elemento,k,b,n)
112         y_convertida3.append(valor)
113
114
115
116
117
118
119
120     plt.figure(figsize=(9,5))
121     plt.plot(x_convertida,y,"r",x_convertida,y_convertida3,"*") #Cambia el color de l
122     plt.title("Datos reales vs datos modelados MIDILLI KUCUK")
123     plt.ylabel('Contenido de Humedad')
124     plt.xlabel('Tiempo (minutos)')
125     plt.legend(["contenido de humedad real","contenido de humedad modelado"],loc =1)
126     #Si no se colocan corchetes no se muestra el mensaje entero
127     plt.show()
128
```

Imagen 77

Posteriormente se define otra función en la línea 96 (imagen 77) la cual requiere para su funcionamiento las variables retornadas en la función del comienzo.

Se definen el valor de las variables encontradas luego del ajuste para ser utilizadas en la nueva grafica de datos que incluyan las constantes.

108) se define una lista vacía llamada “y_convertida3”

109) Se declara un ciclo for para recorrer toda la lista de los elementos que corresponden al eje x, luego este valor de x junto a las constantes es evaluado en la ecuación de Midilli Kucuk definida anteriormente, guardando los valores en la lista y_convertida3”.

120) Se declara las dimensiones de la figura

121) se grafican con el comando plt.plot, pasándole los argumentos que queremos graficar. En este caso se le pasan los valores del tiempo x, y la variable y los cuales serán graficados de color rojo. Luego se grafica los valores de x junto con los valores obtenidos al reemplazar los datos obtenidos en la ecuación . Esta segunda grafica estará contenida en la misma figura y su distinción será un “*” que por defecto es de color azul.

122) se le coloca el nombre a la figura

123 y 124) se les colocan los nombres a los ejes.

125) se le coloca la leyenda a la figura dentro de corchetes esta vez para que queden separados.

127) muestra la figura.

Referencias

1) Python.org. El tutorial de Python. Python.org

<https://docs.python.org/es/3/tutorial/>

2)Alvares. V,. Soto.A.(2020). Introducción a la programación en Python.

<https://adriansoto.cl/pdf/pythonbook.pdf>

3) Computer science Academy. (2019). Python for data Science.

Consulta Libro físico

4) Universidad Católica de Chile. introducción a la programación en Python 1: aprendiendo a programar en Python.

Coursera.org

5)Página oficial de Pandas.

https://pandas.pydata.org/docs/user_guide/indexing.html

6)Página oficial de numpy.

<https://numpy.org/>

7)documentación referente a Scipy

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

8)documentación de matplotlib

<https://matplotlib.org/>

.