1. La instalación es siguiente, siguiente y el ejecutable se descarga desde https://www.python.org/downloads/.

2. Se puede ejecutar con Sublime Text, desde la consola de Windows, desde la consola de Python y desde https://jupyter.org/try.

3. **Variables**:

Ejemplo:

Numero = 5

Cadena = “hola mundo”

- Cuando son cadenas de caracteres se coloca en comillas simples.
- Comienza con letra o guion bajo.
- Es sensible a mayúscula.
- Se concatena con el +.
- Para saber el tipo de la variable se usa la función **type()**.
- Los decimales se dividen por coma.
- Para convertir una variable a string se usa la función **str()**.
- Para convertir una variable a entero se usa la función **int()**.
- Para convertir una variable a decimal se usa la función **float()**.
- Es necesario convertir un número a string si se quiere concatenar con una cadena.
- Los índices comienzan en 0 y -1 si se quiere de forma inversa. Ejemplo en la variable Cadena anterior Cadena [5] sería la letra la “m” y Cadena [-1] es “o”.
- Para mostrar solo parte de una cadena se hace, por ejemplo, Cadena [2:7] que es “la mu”, ya que es posición inicial: posición final menos 1., otro ejemplo Cadena [2:] que toma desde la posición 2 al final.
- Para saber la longitud de una cadena se usa la función **len()**.
- Para mostrar una cadena a mayúscula se usa la función **variable.upper()**. Se muestra, pero la variable va a continuar siendo minúscula.

- Similar con **variable.lower()**.
- Para crear una lista con las palabras de una cadena se usa **cadena.split()**, también puede ser, por ejemplo: **cadena.split(‘,’)**.
- En un **print** la coma es un espacio.
- Para imprimir se usa **print()**, por ejemplo

  print("Buenos días " + nombre)

- Otra forma de imprimir es con la función **.format()**, por ejemplo,

  print("Buenos días {}, feliz {} cumpleaños.".format(nombre,edad))

- Para formatear un número y decirle que solo muestre 3 decimales se puede hacer, por ejemplo:

  resultado = 10/3
  print("El resultado es {r:1.3f}".format(r=resultado))

- Otra forma de imprimir.

  print(f"Buenos días {nombre}, feliz {edad} cumpleaños.")

4. **Tomar texto por pantalla**

- Se usa la función **input()**. Ejemplo,

  print("Introduzca un nombre: ")
  entrada = input()
  print("Hola " + entrada)

5. Operadores

- Aritméticos (+, -, *, /, %, **, //).
- Asignación (=, +=, -=, *=, /=, **=).
- Comparación (==, !=, >, <, >=, <=).
- Lógicos (and, or, not).
- Identidad (is, is not), verifica si 2 objetos son iguales o no.
- Pertenencia (in, not in), verifica si un elemento esta dentro de un objeto.

6. Colección de datos

- Listas:
    - La forma es, por ejemplo, colores = [“rojo”, “amarillo”, “verde”].
    - Se puede modificar con su índice, que comienza en 0, por ejemplo, colores[0] = “azul”.
    - Con la función **len()** se calcula el largo de la lista.
    - Con la función **append()** se puede agregar un nuevo elemento a la lista, por ejemplo, colores.append(“naranja”).
    - Con la función **remove()** se puede eliminar un elemento, por ejemplo, colores.remove(“rojo”).
    - Para vaciar una lista se usa la función **clear()**, por ejemplo colores.clear().

- Tuplas:
    - Colección de elementos ordenados que no se pueden eliminar.
    - Se coloca entre paréntesis, por ejemplo, tupla_colores = (“rojo, “verde”, “amarillo”).
    - Se trabaja igual que una lista, pero no se puede modificar, eliminar, ni agregar nuevos elementos.

- Conjuntos:
    - Colección de elementos desordenados, es decir, no posee índice.
    - Se identifica con llaves, por ejemplo, conjunto_colores = {“rojo”, “verde”, “azul”}.
    - Con la función **len()** se puede saber su tamaño.
    - Se puede agregar elementos, por ejemplo, conjunto_colores.**add**(“amarillo”).
    - Se puede remover un elemento, por ejemplo, conjunto_colores.**remove**(“verde”).

- Diccionarios:
    - Colección de elementos indexados y no ordenados.
    - Se escribe entre llaves y posee clave valor, por ejemplo, diccionario_colores = {“red”:”rojo”, “blue”:”azul”, “yellow”:”amarillo”}.
    - Se accede a cada elemento con su clave, por ejemplo, diccionario_colores[“red”].
    - Para agregar un nuevo valor, por ejemplo, diccionario_colores[“black”] = “negro”.
    - Para eliminar un valor se hace, por ejemplo, diccionario_colores.**pop**(“yellow”).
    - Otra forma de eliminar es con la función **del()**, por ejemplo, del(diccionario_colores[“black”]).
    - Para imprimir clave y valor se hace, por ejemplo,

```
For clave,valor in diccionario_colores.items():
        Print(clave,valor)
```

7. Bucles y condiciones

- Condiciones if, elif, else:

If(condiciones1):
<operaciones1>
Elif(condiciones2):
<operaciones2>
Else:
<operaciones3>

- Blucle for:

  - Ejemplos:

```
colores = ["rojo", "verde", "azul"]
for color in colores:
    print(color)

cadena = "Hola mundo"
for caracter in cadena:
    print(caracter)

for numero in range(8):
    print(numero)

for numero in range(3,8):
    print(numero)

for numero in range(3,8,2):
    print(numero)
```

```
for numero in range(3,8,4):
    print(numero)

for numero in range(10):
    if(numero == 5):
        break
    print(numero)

for numero1 in range(4):
    for numero2 in range(3):
        print(numero1, numero2)
```

- Para generar un rango de valores se usa la función **range(N°)**, por ejemplo si quiero un rango que va del 0 al 7 se hace range(8), y para recorrerlo:

```
For numero in range(8):
        Print(numero)
```

- El **range** también puede ser de esta forma **range(3,8)**, generando un rango del 3 al 7.
- Para que recorra el **range**, por ejemplo del 3 al 7 de dos en dos se hace **range(3,8,2)**.
- El **break** también funciona para parar las iteraciones.
- El **continue** también funciona para parar la iteración actual.
-

- Bucle while:

    - Por ejemplo:

```
valor = 1
fin = 10
while(valor < fin):
    print(valor)
    valor +=1
```

    - También funciona el **break** y el **continue**.

8. Clases, objetos y funciones

- Clases y objetos:
    - Una clase es como un constructor de objetos.
    - Un objeto es la instancia a una clase.

```
class ClaseSilla:
    color = 'blanco'
    precio = 100
objetoSilla1 = ClaseSilla()
objetoSilla1.color
objetoSilla1.precio
```

    - Otro ejemplo:

```
class Persona:
   # Método constructor, donde self indica que ocupa
   # las propiedades de la clase
   def __init__(self, nombre, edad):
      self.nombre = nombre
      self.edad = edad
   # Método que devuelve un saludo
   def saludar(self):
      print('Hola, me llamo {} y tengo {} años.'.format(self.nombre, self.edad))

# Se instancia la clase
persona1 = Persona('Juan',33)
```

- Otro ejemplo:

```
class Coche:
    def __init__(self, marca, color, combustible, cilindrada):
        self.marca = marca
        self.color = color
        self.combustible = combustible
        self.cilindrada = cilindrada
    def mostrar_caracteristicas(self):
        print("{}, {}, {}, {}".format(self.marca, self.color, self.combustible, self.cilindrada))

coche1 = Coche('Opel', 'Rojo', 'Gasolina', '1.6')
coche1.mostrar_caracteristicas()
```

- Funciones
    - Es un código que se ejecuta cuando es llamado.
    - Se declara con la variable **def**.
    - Se llama usando el nombre entre paréntesis. Por ejemplo:

```
def saludar():
    print("Buenos días")

saludar()
```

- También se le puede pasar parámetros. Por ejemplo:

```
def saludar(nombre):
    print("Buenos días {}".format(nombre))
nombre = 'Antonio'
saludar(nombre)
```

- Primero se declaran las funciones y/o clase y luego se usan. Ejemplo:

```
numero1 = 10
numero2 = 20
def suma(numero1, numero2):
    suma = numero1 + numero2
    return suma
resultado = suma(numero1, numero2)
print("El resultado de la suma de {} y {} es {}".format(numero1, numero2, resultado))
```

- Para pasar por referencia se hace, por ejemplo:

```
colores = ['rojo', 'verde', 'azul']

def anadirColor(colores, color):
    colores.append(color)

color = 'negro'
anadirColor(colores, color)
```

- Funciones lambda
  - Es una función pequeña que entrega un resultado.
  - Por ejemplo,

```
resultado = lambda numero : numero + 30
resultado(10)
```

  - Ejemplo 2,

```
resultado2 = lambda numero1, numero2 : numero1 + numero2
resultado2(10,30)
```

  - Ejemplo 3,

```
media = lambda nota1, nota2, nota3 : (nota1 + nota2 + nota3) / 3
media(10, 20, 30)
```

9. Módulos

- Es un fichero que contiene un conjunto de instrucciones que se pueden usar.
- Por ejemplo:

```
# Dentro de un archivo modulo.py colocamos:
def saludar(nombre):
        print("Hola, soy " + nombre)
def despedirse(nombre):
        print(“Adiós “ + nombre
#En otro archivo se llama al módulo, por ejemplo, colocamos:
import modulo
modulo.saludar('Juan')
```

- Si quiero solo usar una sola función de un módulo, se hace, por ejemplo:

```
from modulo import despedirse
#En este caso se usa directo la función y no se necesita anteponer modulo.
despedirse('Pedro')
```

- Se puede usar alias para utilizar las funciones que están en el módulo con otro nombre, por ejemplo:

```
from modulo import despedirse as adios
adios('Pedro')
```

- Para instalar módulos nuevos se usa **Pip** que es un gestor de paquetes y módulos para phyton. Dentro de la consola se puede hacer:

    - Versión:

      # pip --version

    - Listar módulos:

      # pip list

    - Instalar un módulo:

      # pip Install <modulo>

    - Ejemplo con el modulo camelcase que permite pasar a mayúscula:

      # pip
      import camelcase
      camel = camelcase.CamelCase()
      texto = "mi nombre es antonio"
      print(camel.hump(texto))

      El resultado es **Mi Nombre Es Antonio**

    - Desinstalar un módulo:

      # pip uninstrall camelcase

- Ejemplo:

```
#Modulo1
class Coche:
    def __init__(self, marca, color, combustible, cilindrada):
        self.marca = marca
        self.color = color
        self.combustible = combustible
        self.cilindrada = cilindrada
    def mostrar_caracteristicas(self):
        print("{}, {}, {}, {}".format(self.marca, self.color,
self.combustible, self.cilindrada))

media = lambda nota1, nota2, nota3 : (nota1 + nota2 + nota3) / 3

#Programa1
import modulo1
coche1 = modulo1.Coche('Opel', 'Rojo', 'Gasolina', '1.6')
coche1.mostrar_caracteristicas()
nota1 = 5.0
nota2 = 6.1
nota3 = 4.5
promedioNotas = modulo1.media(nota1, nota2, nota3)
print("El promedio entre las nota1 = {}, nota2 = {} y nota3 = {} es
{}".format(nota1, nota2, nota3, promedioNotas))
```

10. **Ficheros**:

- Leer ficheros:
    - Por ejemplo, para un fichero de texto:

```
fichero = open("ficherotexto.txt","rt")
datos_fichero = fichero.read()
print(datos_fichero)
```

- Grabar ficheros:
    - Por ejemplo, crear y agregar información a un fichero de texto nuevo:

```
fichero = open("fichero_para_grabar.txt", "wt")
texto_de_fichero = "Hola, esta es la línea que vamos a grabar en el fichero de texto"
fichero.write(texto_de_fichero)
fichero.close()
```

- Incluir datos de un fichero:
    - Por ejemplo, agregar información a un fichero existente:

```
fichero = open("ficherotexto.txt", "at")
cadena_para_incluir = "\nEsta es la tercera fila del fichero"
fichero.write(cadena_para_incluir)
fichero.close()
```

- Borrar un fichero:

```
import os
os.remove("ficherotexto.txt")
```

- Ejemplo:

```
#moduloficherosdos.py
class Fichero:
        def __init__(self, nombre):
                self.nombre = nombre

        def grabar_fichero(self, texto):
                fichero = open(self.nombre, "wt")
                fichero.write(texto)
                fichero.close()

        def incluir_fichero(self, texto):
                fichero = open(self.nombre, "at")
                fichero.write(texto)
                fichero.close()

        def leer_fichero(self):
                fichero = open(self.nombre, "rt")
                texto = fichero.read()
                return texto
```

```
# programa2.py
import modulficherosdos

nombre_fichero = "ficherin.txt"

fichero = modulficherosdos.Fichero(nombre_fichero)

texto = 'Esta es la primera fila del fichero.\nEsta es la segunda fila del fichero.'
fichero.grabar_fichero(texto)

texto = '\nEsta es la tercera fila del fichero'
fichero.incluir_fichero(texto)

texto_leido = fichero.leer_fichero()
print(texto_leido)
```

11. Ficheros binarios

- Grabar fichero binario:
    - Ejemplo:

```
import pickle
lista_colores = ["azul", "verde", "rojo", "amarillo"]
fichero = open("ficherocolores.pckl", "wb")
pickle.dump(lista_colores, fichero)
fichero.close()
```

- Leer fichero binario:
    - Ejemplo:

```
import pickle
fichero = open("ficherocolores.pckl", "rb")
lista_leida_fichero = pickle.load(fichero)
print(lista_leida_fichero)
```

12. Gestión de errores

- Se usa **try**, **except**, **else** y **finally**.
- Ejemplos:

1)

```
try:
    numero1 = 5
    numero2 = 0
    division = numero1 / numero2
    print(division)
except:
    print("Ha habido un error")
```

2)

```
try:
        numero1 = 5
        numero2 = 0
        division = numero1 / numero2
        print(division)
except ZeroDivisionError:
        print("Error, no se puede dividir por cero.")
except:
        print("Ha habido un error")
```

3)

```
try:
        numero1 = 5
        numero2 = 1
        division = numero1 / numero2
        print(division)
except ZeroDivisionError:
        print("Error, no se puede dividir por cero.")
except:
        print("Ha habido un error")
else:
        print("La división funciono correctamente.")
```

4)

```
try:
        numero1 = 5
        numero2 = 0
        division = numero1 / numero2
        print(division)
except ZeroDivisionError:
        print("Error, no se puede dividir por cero.")
except:
        print("Ha habido un error")
else:
        print("La división funciono correctamente.")
finally:
        print("Esta prueba del try se ha acabado")
```

5)

```
def operaciones(numero1, numero2, numero3):
        try:
                resultado = numero1 / (numero2-numero3)
                print(resultado)
        except ZeroDivisionError:
                print("Error, no se puede dividir por cero. Los últimos dos
número debe ser diferentes")
        else:
                print("El resultado obtenido es correcto.")
        finally:
                print("Ejercicio finalizado.")

numero1 = 5
numero2 = 3
numero3 = 3
```

```
resultado = operaciones(numero1, numero2, numero3)
print(resultado)
```

13. Expresiones regulares, JSON, fecha y hora

- Expresiones regulares:
    - Search: Realiza una búsqueda de un patrón en una cadena.

Ejemplos:

1)

```
texto = "Hola, mi nombre es Antonio"
import re
resultado = re.search("nombre", texto)
if(resultado):
        print("Cadena encontrada: {}".format(resultado))
else:
    print("Cadena NO encontrada")
```

2) Con el $ busca al final y con el ^ busca al principio.

```
texto = "Hola, mi nombre es Antonio"
import re
resultado = re.search("Antonio$", texto)
```

```
if(resultado):
    print("Cadena encontrada: {}".format(resultado))
else:
    print("Cadena NO encontrada")

texto = "Hola, mi nombre es Antonio"
import re
resultado = re.search("^Hola", texto)
if(resultado):
    print("Cadena encontrada: {}".format(resultado))
else:
    print("Cadena NO encontrada")
```

3) Con el .* se puede indicar que existen mas caracteres entre lo que se esta buscando.

```
texto = "Hola, mi nombre es Antonio"
import re
resultado = re.search("mi.*es", texto)
if(resultado):
    print("Cadena encontrada: {}".format(resultado))
else:
    print("Cadena NO encontrada")
```

4) Con los métodos **start** y **end** se puede obtener la posición del hallazgo. Ejemplo:

```
import re
def buscar(texto, palabra_a_buscar):
    resultado = re.search(palabra_a_buscar, texto)
    return resultado
texto = "Esto es una frase de pruebas para hacer busquedas"
palabra_a_buscar = "frase"
resultado = buscar(texto, palabra_a_buscar)
if(resultado):
    posicion_inicial = resultado.start()
    posicion_final = resultado.end()
    print("Frase encontrada desde la posición {} hasta la {}".format(posicion_inicial, posicion_final))
else:
    print("Frase no encontrada")
```

- Findall: Busca todas las ocurrencias en una cadena.

Ejemplos:

1)

```
texto = """
El coche de luis es rojo,
el coche de Antonio es blanco,
y el coche de Maria es rojo
"""
import re
print(re.findall("coche.*rojo", texto))
```

- Split: Divide una cadena a partir de un patrón.

Ejemplos:

1)

```
texto = "La silla es blanca y vale 80"
import re
resultado = re.split("\s", texto)
print(resultado)
```

- Sub: Substituye todas las coincidencias de una cadena.

Ejemplos:

1)

```
texto = "La silla es blanca y vale 80"
import re
resultado = re.sub("blanca", "roja", texto)
print(resultado)
```

- JSON: Forma de almacenar y comparar datos.

```
producto1 = {"nombre":"silla", "color":"blanco", "precio":80}
import json
#Genera el JSON
estructura_json = json.dumps(producto1)
print(estructura_json)
#Pasa a JSON a una estructura normal
productos2 = json.loads(estructura_json)
print(productos2)
```

- Fecha y hora:

```
from datetime import datetime
fechayhora = datetime.now()
print(fechayhora)
ano = fechayhora.year
mes = fechayhora.month
dia = fechayhora.day
hora = fechayhora.hour
minutos = fechayhora.minute
segundos = fechayhora.second
microsegundos = fechayhora.microsecond
print("La hora es {}:{}:{}".format(hora,minutos,segundos))
print("La fecha es {}/{}/{}".format(dia,mes,ano))
```

14. Bases de datos

- Crear base de datos:

```
import sqlite3
conexion = sqlite3.connect("basedatos1.db")
conexion.close()
```

- Crear tabla:

    - El método **cursor** que crea un objeto que permite ejecutar sentencias en la base de datos.
    - El método **execute** permite ejecutar cualquier sentencia SQL.
    - El método **commit** le indica al sistema que la sentencia esta correcta y que la vamos a mantener.

```
import sqlite3

conexion = sqlite3.connect("basedatos1.db")

conexion.cursor()

cursor = conexion.cursor()

cursor.execute("CREATE TABLE PERSONAS (nombre TEXT, apellido1 TEXT, apellido2 TEXT, edad INTEGER)")

conexion.commit()

conexion.close()
```

- Insertar registro:

```
import sqlite3
conexion = sqlite3.connect("../basedatos1.db")
cursor = conexion.cursor()
cursor.execute("INSERT INTO PERSONAS VALUES ('Antonio','Perez','Gomez',35)")
conexion.commit()
conexion.close()
```

- Insertar varios registros:

    - Se usa el método **executemany**.

```
import sqlite3

conexion = sqlite3.connect("../basedatos1.db")

cursor = conexion.cursor()

lista_personas = [

    ('Pedro','Rodriguez','Perez',26),

    ('Maria','Lopez','Gomez',45),

    ('Luis','Gonzalez','Perez',46)

]

cursor.executemany("INSERT INTO PERSONAS VALUES (?,?,?,?)", lista_personas)

conexion.commit()

conexion.close()
```

- Consultar registros:
    - Se usa el método **fetchall** que recoge las filas y columnas.

Ejemplo 1)

```
import sqlite3
conexion = sqlite3.connect("../basedatos1.db")
cursor = conexion.cursor()
cursor.execute("SELECT * FROM PERSONAS")
personas = cursor.fetchall()
for persona in personas:
    print(persona)
conexion.close()
```

Ejemplo 2)

```
import sqlite3
conexion = sqlite3.connect("../basedatos1.db")
cursor = conexion.cursor()
cursor.execute("SELECT * FROM PERSONAS WHERE edad > 40")
personas_seleccionadas = cursor.fetchall()
for persona in personas_seleccionadas:
    print(persona)
conexion.close()
```

Ejemplo 3)

```
import sqlite3

conexion = sqlite3.connect("../basedatos1.db")

cursor = conexion.cursor()

cursor.execute("SELECT * FROM PERSONAS ORDER BY edad")

lista_personas_ordenada = cursor.fetchall()

for persona in lista_personas_ordenada:

    print(persona)

conexion.close()
```

- Borrar datos:

```
import sqlite3
conexion = sqlite3.connect("../basedatos1.db")
cursor = conexion.cursor()
cursor.execute("DELETE FROM PERSONAS WHERE nombre = 'Luis'")
conexion.commit()
conexion.close()
```

- Actualizar datos:

```
import sqlite3
conexion = sqlite3.connect("../basedatos1.db")
cursor = conexion.cursor()
cursor.execute("UPDATE PERSONAS SET edad = 30 WHERE nombre = 'Antonio'")
conexion.commit()
conexion.close()
```

- Ejemplo completo:

```
import sqlite3

conexion = sqlite3.connect("basededatos.db")

cursor = conexion.cursor()

cursor.execute("CREATE TABLE PRODUCTOS (id INTEGER, nombre TEXT, precio INTEGER)")

conexion.commit()


lista_productos = [

(1,'Impresora',300),

(2,'Ratón',20),

(3,'Ordenador',600)

]

cursor.executemany("INSERT INTO PRODUCTOS VALUES (?,?,?)", lista_productos)
```

```
conexion.commit()


cursor.execute("SELECT * FROM PRODUCTOS")
productos = cursor.fetchall()
for producto in productos:
    print(producto)


conexion.close()
```

15. Interfaz gráfica con el módulo tkinter

- Introducción:
    - El módulo **tkinter** tiene una serie de componentes llamados Widgets que son componentes gráficos.
    - **Tk**: componente que contiene el resto de los componentes, es decir, la raíz.
    - **Frame**: es un marco que puede contener otros widgets y que tiene un tamaño propio.
    - **Label**: etiqueta donde se puede insertar texto.
    - **Entry**: campo de texto sencillo para escribir un texto corto, como un nombre en un formulario.
    - **Text**: campo de texto grande para escribir varias líneas de texto, como un campo comentario.
    - **Button**: Botón.
    - **RadioButton**: Botón de opciones.
    - **CheckButton**: Botón de marcado.

- **Menu**: componente para crear los menús de las aplicaciones.
- **Dialogs**: componente para crear ventanas emergentes como alertas o confirmaciones.

- Componente raíz (tk):

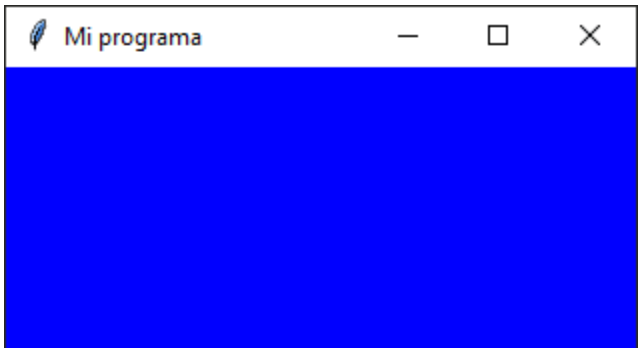
  - El método **mainloop** provoca que se ejecute todo el rato.

```
import tkinter
raiz = tkinter.Tk()
raiz.title("Mi programa")
raiz.mainloop()
```

- Componente frame:

  - El método **pack** hace que se muestre por pantalla.
  - **Bg** es el color de fondo, **width** el ancho y **height** el alto.

```
import tkinter
raiz = tkinter.Tk()
raiz.title("Mi programa")
frame = tkinter.Frame(raiz)
frame.config(bg="blue",width=400,height=300)
frame.pack()
raiz.mainloop()
```



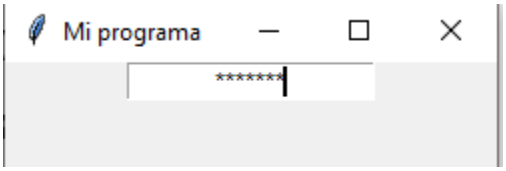
- Componente label:
    - **Fg** el color de la letra, **bg** el color del fondo de la letra y **font** es la fuente y tamaño.

```
import tkinter
raiz = tkinter.Tk()
raiz.title("Mi programa")
texto = "Hola mundo"
etiqueta = tkinter.Label(raiz,text=texto)
etiqueta.config(fg="green",bg="lightgrey",font=("Cortana",30))
etiqueta.pack()
raiz.mainloop()
```

- Componente entry:

    - Permite ingresar información por teclado.
    - **Show** permite convertir el texto en un campo de contraseña.

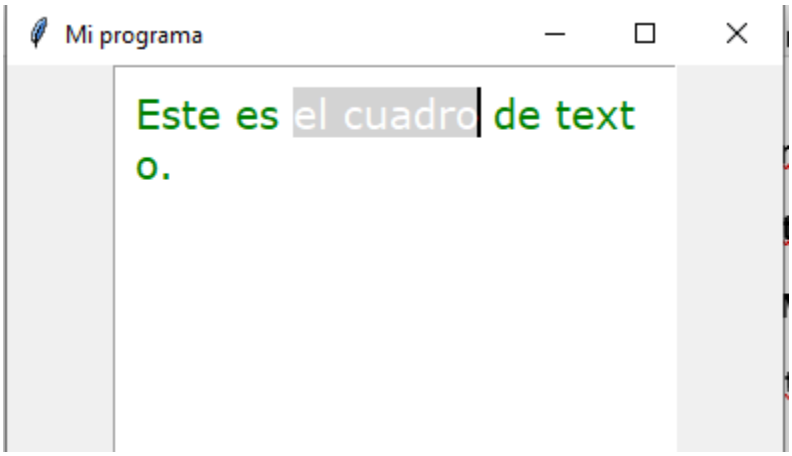
```
import tkinter

raiz = tkinter.Tk()

raiz.title("Mi programa")

entrada = tkinter.Entry(raiz)

entrada.config(justify="center",show="*")

entrada.pack()

raiz.mainloop()
```



- Componente text:

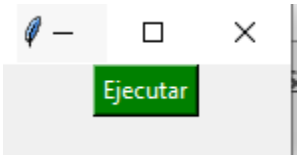
    - **Padx** es el espaciado en el eje X, **Pady** es el espaciado en el eje Y y **selectbackground** es el color del seleccionado.

```
import tkinter

raiz = tkinter.Tk()

raiz.title("Mi programa")

entrada = tkinter.Text(raiz)

entrada.config(width=20,height=10,font=("Verdana",15),padx=10,pady=10,fg="green",selectbackground="lightgrey")

entrada.pack()

raiz.mainloop()
```



- Componente button:

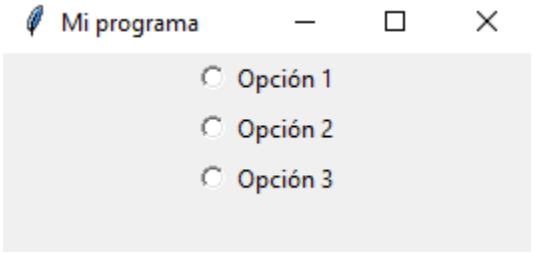
```
import tkinter

raiz = tkinter.Tk()
raiz.title("Mi programa")

def accion():
        print("Hola mundo")

boton = tkinter.Button(raiz, text="Ejecutar", command=accion)
boton.config(fg="white", bg="green")
boton.pack()

raiz.mainloop()
```



- Componente radiobutton:

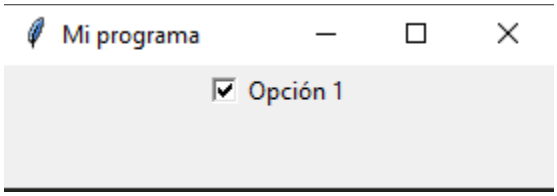
```
import tkinter
raiz = tkinter.Tk()
raiz.title("Mi programa")


def seleccionar():
        print("La opción seleccionada es {}".format(opcion.get()))
```

```
opcion = tkinter.IntVar()

botonradio1 = tkinter.Radiobutton(raiz, text="Opción 1", variable=opcion, value=1, command=seleccionar)

botonradio1.pack()

botonradio2 = tkinter.Radiobutton(raiz, text="Opción 2", variable=opcion, value=2, command=seleccionar)

botonradio2.pack()

botonradio3 = tkinter.Radiobutton(raiz, text="Opción 3", variable=opcion, value=3, command=seleccionar)

botonradio3.pack()

raiz.mainloop()
```



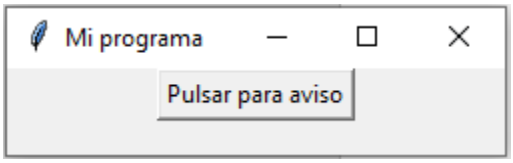
- Componente checkbutton:

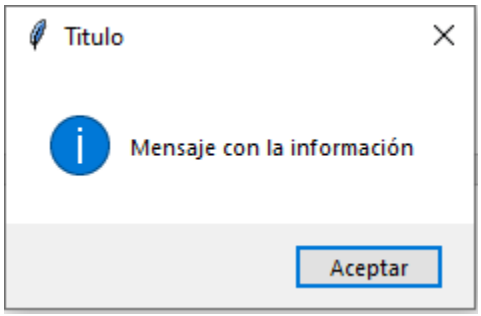
```
import tkinter

raiz = tkinter.Tk()

raiz.title("Mi programa")


def verificar():
        valor = check1.get()
        if(valor == 1):
                print("El check esta activado")
        else:
                print("El check esta desactivado")
check1 = tkinter.IntVar()

boton1 = tkinter.Checkbutton(raiz, text="Opción 1", variable=check1, onvalue=1, offvalue=0, command=verificar)

boton1.pack()

raiz.mainloop()
```



- Componente messagebox:
    - Desde tkinter hay que importar la librería **messagebox**.

```
import tkinter
from tkinter import messagebox

raiz=tkinter.Tk()
raiz.title("Mi programa")

def avisar():
    tkinter.messagebox.showinfo("Titulo", "Mensaje con la información")

boton = tkinter.Button(raiz, text="Pulsar para aviso", command=avisar)
boton.pack()

raiz.mainloop()
```

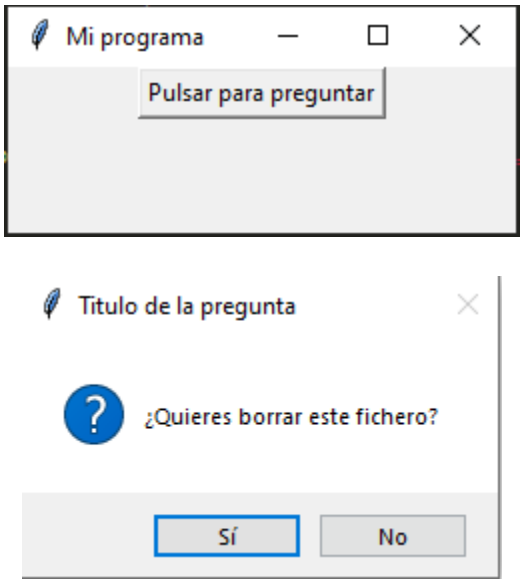




- Para realizar una consulta con un messagebox:

```
import tkinter

from tkinter import messagebox


raiz = tkinter.Tk()

raiz.title("Mi programa")


def preguntar():

    resultado = tkinter.messagebox.askquestion("Titulo de la pregunta", "¿Quieres borrar este fichero?")

    if(resultado == "yes"):

            print("Si, quiero borrar el fichero")

    else:

            print("No, no quiero borrar el fichero")

boton = tkinter.Button(raiz, text="Pulsar para preguntar", command=preguntar)

boton.pack()
```

raiz.mainloop()



- Componente filedialog:
    - Desde tkinter hay que importar la librería **filedialog**.

import tkinter

from tkinter import filedialog

raiz = tkinter.Tk()

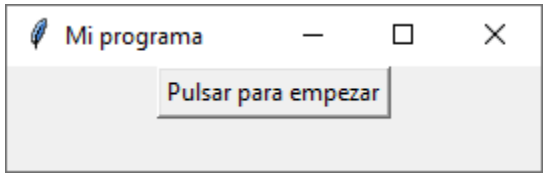
raiz.title("Mi programa")

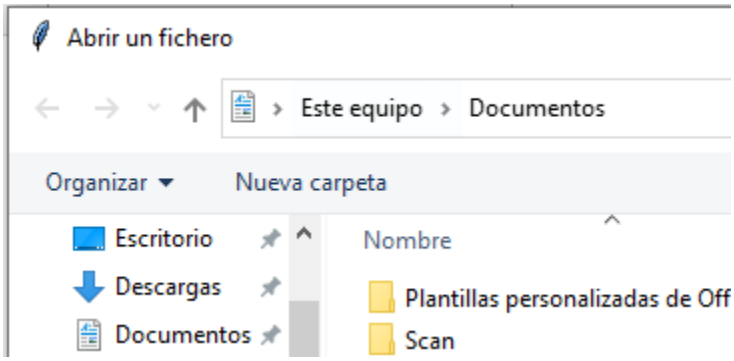
def abrirfichero():

rutafichero = filedialog.askopenfilename(title="Abrir un fichero")

```
    print(rutafichero)
```

```
boton = tkinter.Button(raiz, text="Pulsar para empezar", command=abrirfichero)
boton.pack()

raiz.mainloop()
```





- Ejercicio – Calculadora:

```
#Se importan todas las librerias de tkinter
from tkinter import *

#Generamos el elemento raiz (ventana)
ventana = Tk()
ventana.title("Calculadora")
#Indicamos el tamaño
ventana.geometry("350x600")
#Que no se pueda redimensionar
ventana.resizable(False,False)
#Color de la ventana
ventana.configure(background="gray")

#Variables
operacion = ""
resultado = StringVar()
```

```
#Funciones
def borrar():
    global operacion #Variable global
    global resultado #Variable global
    operacion = "" #Vacia variable
    pantalla.delete(0, END) #Borra el Entry
def pulsar(valor):
    global operacion #Variable global
    global resultado #Variable global
    operacion = operacion + str(valor) #Concatena
    pantalla.delete(0, END) #Borra la pantalla
    pantalla.insert(0, operacion) #Muestra lo concatenado
def calcular():
    global operacion #Variable global
    global resultado #Variable global
    try:
        #Evalua lo concatenado
        resultado = str(eval(operacion))
    except:
        resultado = "Error"
    finally:
        pantalla.delete(0, END)
        pantalla.insert(0, resultado)
```

```
#Display de los resultados
pantalla = Entry(ventana, font=("Arial", 20, "bold"), width=17, borderwidth=2)
#Cantidad de filas y columnas que ocupa el display junto
#con el espaciado en X e Y
pantalla.grid(row=0, column=0, columnspan=3, pady=10, padx=5)
#Botón de reiniciar operación
boton_reset = Button(ventana, text="AC", width=8, height=2, command=lambda:borrar())
#Posición del botón
boton_reset.grid(row=0, column=3, pady=10, padx=5)
```

```
#Botones de la primera fila
ancho = 8
alto = 3

boton1 = Button(ventana, text="1", width=ancho, height=alto, command=lambda:pulsar("1"))
boton1.grid(row=1, column=0, padx=5, pady=5)

boton2 = Button(ventana, text="2", width=ancho, height=alto, command=lambda:pulsar("2"))
boton2.grid(row=1, column=1, padx=5, pady=5)

boton3 = Button(ventana, text="3", width=ancho, height=alto, command=lambda:pulsar("3"))
boton3.grid(row=1, column=2, padx=5, pady=5)

boton4 = Button(ventana, text="4", width=ancho, height=alto, command=lambda:pulsar("4"))
boton4.grid(row=1, column=3, padx=5, pady=5)
```

```
#Botones de la segunda fila
boton5 = Button(ventana, text="5", width=ancho, height=alto, command=lambda:pulsar("5"))
boton5.grid(row=2, column=0, padx=5, pady=5)

boton6 = Button(ventana, text="6", width=ancho, height=alto, command=lambda:pulsar("6"))
boton6.grid(row=2, column=1, padx=5, pady=5)

boton7 = Button(ventana, text="7", width=ancho, height=alto, command=lambda:pulsar("7"))
boton7.grid(row=2, column=2, padx=5, pady=5)

boton8 = Button(ventana, text="8", width=ancho, height=alto, command=lambda:pulsar("8"))
boton8.grid(row=2, column=3, padx=5, pady=5)
```

```
#Botones de la tercera fila
boton9 = Button(ventana, text="9", width=ancho, height=alto, command=lambda:pulsar("9"))
boton9.grid(row=3, column=0, padx=5, pady=5)

boton0 = Button(ventana, text="0", width=ancho, height=alto, command=lambda:pulsar("0"))
boton0.grid(row=3, column=1, padx=5, pady=5)

boton_punto = Button(ventana, text=".", width=ancho, height=alto, command=lambda:pulsar("."))
boton_punto.grid(row=3, column=2, padx=5, pady=5)

boton_suma = Button(ventana, text="+", width=ancho, height=alto, command=lambda:pulsar("+"))
boton_suma.grid(row=3, column=3, padx=5, pady=5)
```

```
#Botones de la cuarta fila
boton_resta = Button(ventana, text="-", width=ancho, height=alto, command=lambda:pulsar("-"))
boton_resta.grid(row=4, column=0, padx=5, pady=5)

boton_multi = Button(ventana, text="*", width=ancho, height=alto, command=lambda:pulsar("*"))
boton_multi.grid(row=4, column=1, padx=5, pady=5)

boton_divi = Button(ventana, text="/", width=ancho, height=alto, command=lambda:pulsar("/"))
boton_divi.grid(row=4, column=2, padx=5, pady=5)

boton_igual = Button(ventana, text="=", width=ancho, height=alto, command=lambda:calcular())
boton_igual.grid(row=4, column=3, padx=5, pady=5)

#Que el programa siempre se ejecute
ventana.mainloop()
```

16. Generar documentación automáticamente:

- Docstrings: Son cadenas para generar automáticamente documentación del código.

    - Los comentarios comienzan y terminan con 3 comillas dobles.
    - Para mostrar los comentarios se usa **help(nombre_función)**.

```
def saludar(nombre):
    """
    Esto será un comentario de la función saludar.
    Esta función recibirá como parametro una cadena con el nombre e
    imprimirá por pantalla un saludo con el nombre concatenado
    """
    print("Buenos días " + nombre)

saludar("Antonio")
help(saludar)
```

    - Help se puede usar para cualquier función, por ejemplo, **help(print)**.

```python
class Saludos:
    """
    Esta es la documentación de la clase Saludos
    Tendrá dos funciones buenos_dias y adios
    y ambas será necesario pasarle un parametro con el nombre de la persona
    """
    def buenos_dias(self, nombre):
        """ Sirve para dar los buenos días a un nombre pasado por parametro """
        print("Buenos días {}".format(nombre))

    def adios(self, nombre):
        """ Sirve para decir adios a un nombre pasado por parametro """
        print("Adios {}".format(nombre))

saludos = Saludos()
saludos.buenos_dias("Pedro")
saludos.adios("Pedro")
```

- Pydoc: Genera documentación automática desde la consola o terminal de comandos.

    - En Windows hay que ir a la ruta donde se encuentra la librería pydoc.py.
    - Se puede usar para acceder a cualquier documentación, por ejemplo, **pydoc len**.
    - Para generar un fichero HTML se hace con **pydoc -w <ruta del archivo py>**

17. Pruebas automáticas

- Doctest: Genera pruebas dentro de la documentación. Ejemplo,

```python
def sumar(numero1, numero2):
    """
    Esta es la documentación de esta función
    Recibe dos números como parámetros y devuelve la suma
    >>> sumar(4,3)
    7
    """
    return numero1+numero2

resultado = sumar(2,4)
print(resultado)

import doctest
doctest.testmod()
```

```
C:\Users\Manuel Levicoy O\OneDrive - Bogado Ingenieros Consultores SpA\Python\PruebasAutomaticas>python doctest1.py -v
6
Trying:
    sumar(4,3)
Expecting:
    7
ok
1 items had no tests:
    __main__
1 items passed all tests:
   1 tests in __main__.sumar
1 tests in 2 items.
1 passed and 0 failed.
Test passed.
```

Donde:

- Se documenta una función.
- Dentro de la documentación se realiza la prueba comenzando con 3 signos menor que y el resultado de la prueba abajo.

```
>>> sumar(4,3)
7
```

- Para utilizar la función **doctest** hay que incluir al final,

```
import doctest
doctest.testmod()
```

- Desde la consola se ejecuta con **-v** para que las pruebas se realicen,

```
python doctest1.py -v
```

- Unittest: Sirve para crear pruebas dentro del propio código.

```
def multiplicar(numero1, numero2):
    """
    Función que multiplica dos número
    que ayudara a realizar la prueba automática
    """
    return numero1 * numero2

# Llamamos a la función multiplicar
resultado = multiplicar(2,4)
# Imprimimos el resultado
print(resultado)

# Importamos la libreria unittest
import unittest
# Creamos la clase que realiza la prueba
class pruebas(unittest.TestCase):
    # Esta función debe ir
    def test(self):
        # Con el método assertEqual se llama a la
        # función multiplicar, y verifica si el resultado
        # es correcto
        self.assertEqual(multiplicar(2,4),8)
# Llamamos a la clase
if __name__ == '__main__':
    unittest.main()
```

```
C:\Users\Manuel Levicoy O\OneDrive - Bogado Ingenieros Consultores SpA\Py
8
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

18. Funciones avanzadas

- Funciones generadoras: Genera valores sobre la marcha, cada vez que se le pide uno lo entrega. Por ejemplo,

```python
# range es una función generadora
# tambien puede ser range(11)
range(0,11)
for numero in range(0,11):
    print(numero)
# Crearemos una función generadora personalizada
def pares(maximo):
    for numero in range(maximo):
        if(numero % 2 == 0):
            # Yiend es un objeto que devuelve un
            # número a la vez
            yield(numero)
# Probamos la función pares
maximo = 11
for numero in pares(maximo):
    print(numero)
```

- Filter: Filtra resultados a partir de una lista y una función condicional. Y el resultado es otra lista con los elementos filtrados. Por ejemplo,

```python
# Función que devuelve True si el
# número es positivo y False si es
# negativo
def positivo(numero):
    if(numero > 0):
        return True
    else:
        return False
# Generamos una lista de número
numeros = [4, -2, 8, -3, -5, -7, 1, 9]
# Filtramos llamando primero a la función
# condicional enviandole la lista de números
filtro = filter(positivo, numeros)
# Generamos la lista filtrada
resultado = list(filtro)
# Mostramos la lista
print(resultado)
```

- Map: Aplica una función a una lista. Por ejemplo,

```python
# Función multiplicar un número por 2
def multiplicar(numero):
    return numero * 2
# Llamamos a la función
resultado = multiplicar(2)
# Mostramos el resultado
print(resultado)
# Generamos una lista
numeros = [2, 4, 6]
# Usamos la función map, envianole la función y la
# lista a multiplicar, # devolvera un objeto en
# este caso llamado mapeo
mapeo = map(multiplicar, numeros)
# Convertimos el objeto en una lista
resultadoMapeo = list(mapeo)
# Mostramos la lista
print(resultadoMapeo)
```

```
# Se puede hacer en la misma línea
lista_resultado = list(map(multiplicar, numeros))
print(lista_resultado)
```

```
# Se puede usar una función lambda
lista_resultado_dos = list(map(lambda numero: numero * 2, numeros))
print(lista_resultado_dos)
```

19. Módulo numpy

- Creando arrays:

```
# importamos el modulo numpy y la llamamos np
import numpy as np
```

```
# se genera un array de 4 elementos con ceros
np.zeros(4)
print(np.zeros(4))
# se genera un array de 4 elementos con unos
np.ones(4)
print(np.ones(4))
# se genera un arra de 4 elementos que va del 0 al 4
np.arange(5)
print(np.arange(5))
# similar al anterior pero del 0 al 9
np.arange(10)
print(np.arange(10))
# en este caso se pide un arreglo que comience desde el
# 2 hasta el 20 de 3 en 3
np.arange(2,20,3)
print(np.arange(2,20,3))
```

```python
# se puede crear un array a partir de una lista
lista1 = [1, 2, 3, 4]
array1 = np.array(lista1)
print(array1)
```

```python
# se puede crear una array doble (dos dimensiones)
lista1 = [1, 2, 3, 4]
lista2 = [5, 6, 7, 8]
lista_doble = (lista1, lista2)
print(lista_doble)
array_doble = np.array(lista_doble)
print(array_doble)
```

```python
# verificamos la forma del array, en este caso 2,
# 4 columnas
print(array_doble.shape)
# Verificar el tipo de dato del arreglo, en este caso
# int64
print(array_doble.dtype)
```

- Operaciones con array: (+, -, *, /, **)

```python
# importamos el componente numpy
import numpy as np

# generamos un arreglo de 4 elementos
array1 = np.array([1, 2, 3, 4])
print(array1)

# multiplicamos el array1 por 2
print(array1 * 2)

# dividimos el array1 por 2
print(array1 / 2)

# sumamos 4 al array1
print(array1 + 4)

# restamos 1 al array1
print(array1 - 1)

# elevamos por 2 el array1
print(array1 ** 2)

# si se quiere mantener el valor se hace
# por ejemplo
array2 = np.array([1, 2, 3, 4])
print(array2)
array2 = array2 + 2
print(array2)
```

```python
# se puede hacer con array dobles
lista1 = [1,2,3,4]
lista2 = [5,6,7,8]
lista_doble = (lista1, lista2)
array_doble = np.array(lista_doble)
print(array_doble)
print(array_doble.dtype)
print(array_doble.shape)
print(array_doble + 2)
print(array_doble - 3)
print(array_doble * 4)
print(array_doble / 3)
print(array_doble ** 3)
array_doble = array_doble + 6
print(array_doble)
```

- Indexación:

```python
# importamos el modulo
import numpy as np

# generamos un array que va del 0 al 10
# array([0,1,2,3,4,5,6,7,8,9,10])
array = np.arange(0,11)
print(array)

# toma elementos del 0 al 3
# array([0,1,2])
array[0:3]
print(array[0:3])

# toma elementos del 2 al 5
# array([2,3,4])
array[2:5]
print(array[2:5])

# tomar todos los elementos
array[:]
print(array[:])

# genera una copia
array_copia = array.copy()
```

```
# cambiar valores
# array_copia([20 20 20  3  4  5  6  7  8  9 10])
array_copia[0:3] = 20
print(array_copia[:])

# array de varias dimensiones 3 filas x 3 columnas
array2 = np.array(([1,2,3],[4,5,6],[7,8,9]))
print(array2)

# acceder a las filas
print(array2[0])
print(array2[1])
print(array2[2])

# acceder a un elemento de la columna
# se coloca la fila y la columna
# ejemplo el array2[1][0] = 4
print(array2[1][0])
```

- Matrices traspuestas: permite cambiar ordenadamente las filas por las columnas.

```
# importamos el componente numpy
import numpy as np

# generamos un arreglo de 15 elementos
# repartidos entre 3 filas y 5 columnas
# [[ 0  1  2  3  4]
#  [ 5  6  7  8  9]
#  [10 11 12 13 14]]
array = np.arange(15).reshape((3,5))
print(array)

# acceder a la posición 6
print(array[1][1])

# generamos la matriz traspuesta
# [[ 0  5 10]
#  [ 1  6 11]
#  [ 2  7 12]
#  [ 3  8 13]
#  [ 4  9 14]]
array_tras = array.T
print(array_tras)
```

- Entrada y salida con array:

```
import numpy as np

# [0 1 2 3 4 5]
array1 = np.arange(6)
print(array1)

# salvamos el array y lo renombramos
np.save('array1s', array1)
# rescatamos el array
np.load('array1s.npy')
print(np.load('array1s.npy'))

# otro 2 array
# array2 = [0 1 2 3 4 5]
# array3 = [0 1 2 3 4 5 6 7]
array2 = np.arange(6)
array3 = np.arange(8)
print(array2)
print(array3)
```

```
# salvar ambos arrays con la misma variable
# en la coordenada x le pasamos array2 y en
# la coordenada y le pasamos array3.
np.savez('arrays', x=array2, y=array3)
# rescatamos el array, npz porque son varios array
array_recuperado = np.load('arrays.npz')
# si queremos recuperar el arreglo en x
# x = [0 1 2 3 4 5]
print(array_recuperado['x'])
# si queremos recuperar el arreglo en y
# y = [0 1 2 3 4 5 6 7]
print(array_recuperado['y'])

# guardar en un archivo de texto
np.savetxt('mificheroarray.txt', array2, delimiter=',')
# recuperamos el fichero de texto
# [0. 1. 2. 3. 4. 5.]
print(np.loadtxt('mificheroarray.txt', delimiter=','))
```

- Funciones con arreglos:

```python
import numpy as np

array = np.arange(5)
print(array)

# raiz cuadrada de cada elemento
# [0.         1.         1.41421356 1.73205081 2.        ]
print(np.sqrt(array))

# crear array de forma aleatoria
# [0.37149848 0.77784299 0.49200287 0.37921762 0.00463392]
print(np.random.rand(5))

# array a partir de una lista
# [5 6 7 8]
lista = [5, 6, 7, 8]
array2 = np.array(lista)
print(array2)
```

```python
# [6 6 6 6 6] suma los elementos de los array
array = [1, 2, 3, 4, 5]
array2 = [5, 4, 3, 2, 1]
print(np.add(array, array2))

# [5 4 3 4 5] maximo de cada elemento de los array
print(np.maximum(array, array2))
```

Ejemplo 1: Función que retorna un array con números pares de 2 en 2.

```python
import numpy as np


# funcion que retorna array con numeros pares
def pares(inicio, fin):
    if(inicio % 2 == 0):
        # de 2 en 2
        arreglo = np.arange(inicio, fin, 2)
    else:
        inicio = inicio + 1
        arreglo = np.arange(inicio, fin, 2)
    return arreglo


array = pares(1, 30)
array2 = pares(2,40)
print(array)
print(array2)
```

Ejemplo 2:

```python
import numpy as np
# lista que va del 10 al 19
lista_uno = np.arange(10,20)
print(lista_uno)

# lista que va del 50 al 59
lista_dos = np.arange(50,60)
print(lista_dos)

# lista doble
lista_Doble = (lista_uno, lista_dos)
print(lista_Doble)

# matriz 2X10
matriz = np.array(lista_Doble)
print(matriz)
print(matriz.shape)

# multiplicamos cada elemento por 2
matriz_multi = matriz * 2
print(matriz_multi)
```

Ejemplo 3:

```python
import numpy as np

# lista con 30 elementos
lista1 = np.arange(30)
print(lista1)
# primeros 10 elementos
primeros10 = lista1[0:10]
print(primeros10)
print(primeros10.shape)
# últimos 10 elementos
ultimos10 = lista1[-10:]
print(ultimos10)
print(ultimos10.shape)
# recorremos la lista
for numero in ultimos10:
    print(numero)
```

20. Módulo pandas

- Series: es una matriz unidimensional que contiene datos de cualquier tipo.

```
import pandas as pd

# se genera una lista de valores indexados
# 0    3
# 1    5
# 2    7
# 3    9
# dtype: int64
serie1 = pd.Series([3, 5, 7, 9])
print(serie1)
# Ejemplo si quisiera acceder al número 7
print(serie1[2])

# Se puede generar la indexación
asignaturas = ['matematicas', 'historia', 'fisica', 'literatura']
notas = [8, 6, 9, 7]
serie_notas_daniel = pd.Series(notas, index=asignaturas)
print(serie_notas_daniel)
print(serie_notas_daniel['fisica'])
```

```python
# Se pueden hacer condiciones dentro e la serie
print(serie_notas_daniel[serie_notas_daniel >= 8])

# Se le puede poner un nombre a la serie
serie_notas_daniel.name = 'Notas de Daniel'
print(serie_notas_daniel)

# Se le puede poner un nombre a los indices
# Asignaturas de Daniel
# matematicas    8
# historia       6
# fisica         9
# literatura     7
# Name: Notas de Daniel, dtype: int64
serie_notas_daniel.index.name = 'Asignaturas de Daniel'
print(serie_notas_daniel)
```

```python
# Convertir la serie en un diccionario
# {'matematicas': 8, 'historia': 6, 'fisica': 9, 'literatura': 7}
diccionario = serie_notas_daniel.to_dict()
print(diccionario)

# Convertir un diccionario en una serie
serie = pd.Series(diccionario)
print(serie)

# Otra serie, para ana
asignaturas = ['matematicas', 'historia', 'fisica', 'literatura']
notas_ana = [7, 8, 5, 9]
serie_notas_ana = pd.Series(notas_ana, index=asignaturas)
print(serie_notas_ana)
```

```python
# Serie con la media de la clase
# matematicas    7.5
# historia       7.0
# fisica         7.0
# literatura     8.0
# dtype: float64
serie_notas_clase = (serie_notas_daniel + serie_notas_ana) / 2
print(serie_notas_clase)
```

- DataFrames: estructura bidimensional, es decir, en filas y columnas.

```python
import pandas as pd

# Extraemos información de una web
import webbrowser
website = 'https://es.wikipedia.org/wiki/Anexo:Campeones_de_la_NBA'
webbrowser.open(website)

# Copiamos lo del portapapeles (lo que se copio de la web)
dataframe_nba = pd.read_clipboard()
print(dataframe_nba)

# Mostramos el nombre de las columnas
print(dataframe_nba.columns)

# Mostrar una sola columna, con el dataframe indexado
print(dataframe_nba['Campeón del Oeste'])

# Mostrar un datagrame por la indexación
print(dataframe_nba.loc[5])
```

```python
# Mostrar los primeros elementos
print(dataframe_nba.head())
# Puede ser con el valor de los elementos a mostrar
print(dataframe_nba.head(2))

# Dataframe desde un diccionario
lista_asignaturas = ['matematicas', 'historia', 'fisica']
lista_notas = [8, 7, 9]
diccionario = {'Asignaturas':lista_asignaturas, 'Notas':lista_notas}
print(diccionario)
dataframe_notas = pd.DataFrame(diccionario)
print(dataframe_notas)
```

- Índices:

```python
import pandas as pd

# Generamos una serie
lista_valores = [1, 2, 3]
lista_indices = ['a', 'b', 'c']
serie = pd.Series(lista_valores, index=lista_indices)
# Mostramos la serie
print(serie)
# Mostramos los indices
print(serie.index)
# Mostramos el primer indice
# Los indices no se pueden cambiar
print(serie.index[0])

# Indices a través de un dataframe
lista_valores = [[6,7,8], [8,9,5], [6,9,7]]
lista_indices = ['matematicas', 'historia', 'fisica']
lista_nombres = ['Antonio', 'Maria', 'Pedro']
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_nombres)
# Mostramos el dataframe
print(dataframe)
# Mostramos los indices
print(dataframe.index)
# Mostramos un elemento del indice
print(dataframe.index[2])
```

- Eliminar elementos:

```
# Importamos los modulos
import pandas as pd
import numpy as np

# Mostramos un array(4)
# [0 1 2 3]
print(np.arange(4))

# Generamos la serie
serie = pd.Series(np.arange(4), index=['a', 'b', 'c', 'd'])
print(serie)

# Eliminamos un elemento de una serie usando su índice
print(serie.drop('c'))

# Eliminamos a través de un Dataframe
# [[0 1 2], [3 4 5], [6 7 8]]
lista_valores = np.arange(9).reshape(3, 3)
lista_indices = ['a', 'b', 'c']
lista_columnas = ['c1', 'c2', 'c3']
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)
#    c1  c2  c3
# a   0   1   2
# b   3   4   5
# c   6   7   8

# Eliminamos la fila b
print(dataframe.drop('b'))
#    c1  c2  c3
# a   0   1   2
# c   6   7   8
```

```
# Eliminamos la c2
print(dataframe.drop('c2', axis=1))
#    c1  c3
# a   0   2
# b   3   5
# c   6   8
```

- Seleccionar datos en serie:

```python
# Importamos modulo pandas para las series y dataframes
import pandas as pd
# Importamos modulo numpy para los arrays
import numpy as np

# Creamos una serie
lista_Valores = np.arange(3)
lista_indices = ['i1', 'i2', 'i3']
serie = pd.Series(lista_Valores, index=lista_indices)
print(serie)

# Multiplicamos la serie por 2
serie = serie * 2
print(serie)

# Accedemos al valor a través de su index
print(serie['i2'])

# Accedemos al valor a través de su posición
print(serie[1])

# Accedemos a varios valores de la serie
print(serie[0:2])
```

```python
# Acceder a todos los valores de la serie
print(serie[:])

# Acceder al valor con una condición
print(serie[serie > 3])

# Cambiamos el valor
serie['i3'] = 6
print(serie)
```

- Seleccionar datos en DataFrame:

```
import pandas as pd
import numpy as np

# Generamos una lista de 25 valores distribuidos
# en 5 filas y 5 columnas
lista_valores = np.arange(25).reshape(5,5)
# Son 5 filas por ende 5 indices
lista_indices = ['i1', 'i2', 'i3', 'i4', 'i5']
# Generamos las columnas
lista_columnas = ['c1', 'c2', 'c3', 'c4', 'c5']
# Generamos el dataframe
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)

# Mostramos una columna del dataframe
print(dataframe['c2'])

# Mostramos un elemento es decir columna x fila
print(dataframe['c3']['i1'])

# Mostrar varias columnas
print(dataframe[['c2','c5']])

# Mostramos el dataframe donde la c2 sea mayor a 15
print(dataframe[dataframe['c2'] > 15])

# Muestra True o False cuando los valores son mayores que 20
print(dataframe > 20)

# Seleccionamos por indice
print(dataframe.loc['i5'])
```

- Operaciones:

```python
import pandas as pd
import numpy as np

# Con series
serie1 = pd.Series([0, 1, 2], index=['a', 'b', 'c'])
print(serie1)

serie2 = pd.Series([3, 4, 5, 6], index=['a', 'b', 'c', 'd'])
print(serie2)

# Sumamos los valores que tienen el mismo indice, si no existe el
# indice da un valor nulo
# a    3.0
# b    5.0
# c    7.0
# d    NaN
# dtype: float64
print(serie1 + serie2)
```

```python
# Con DataFrame
lista_valores = np.arange(4).reshape(2,2)
lista_indices = list('ab')
lista_columnas = list('12')
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)

lista_valores_2 = np.arange(9).reshape(3,3)
lista_indices_2 = list('abc')
lista_columnas_2 = list('123')
dataframe_2 = pd.DataFrame(lista_valores_2, index=lista_indices_2, columns=lista_columnas_2)
print(dataframe_2)

# Sumar los elementos que coincidan, los demas lo pone nulo
#     1    2   3
# a  0.0  2.0 NaN
# b  5.0  7.0 NaN
# c  NaN  NaN NaN
dataframe_3 = dataframe + dataframe_2
```

```python
# Para solucionar el problema de los Null
# En el dataframe_2, los valores que no existen los rellenamos con ceros
dataframe_4 = dataframe.add(dataframe_2, fill_value=0)
print(dataframe_4)
```

- Ordenación y clasificación:

```
import pandas as pd
import numpy as np

# Lista de valores
lista_valores = range(4)
print(lista_valores)
# Lista de indices
lista_indices = list('CABD')
print(lista_indices)
# Creamos la serie
serie = pd.Series(lista_valores, index=lista_indices)
print(serie)

# Ordenamos alfabeticamente los indices
print(serie.sort_index())

# Ordenamos por los valores
print(serie.sort_values())

# Generamos un ranking
print(serie.rank())
```

```
# Creamos una serie de 10 números aleatorios
# 0    0.677689
# 1    0.361784
# 2    0.052587
# 3    0.901053
# 4    0.621043
# 5    0.221795
# 6    0.170423
# 7    0.629015
# 8    0.193603
# 9    0.788448
# dtype: float64
serie2 = pd.Series(np.random.rand(10))
print(serie2)

# Generamos un ranking de la serie2
print(serie2.rank())

# Ordenamos la serie 2 por valores
print(serie2.sort_values())
```

- Estadísticas:

```python
import pandas as pd
import numpy as np

# Creamos un dataframe
#    1  2  3
# a  1  8  3
# b  5  6  7
array = np.array([[1, 8, 3], [5, 6, 7]])
dataframe = pd.DataFrame(array, index=['a', 'b'], columns=list('123'))
print(dataframe)

# Suma por columnas
# 1     6
# 2    14
# 3    10
print(dataframe.sum())

# suma por filas
# a    12
# b    18
print(dataframe.sum(axis=1))
```

```python
# Valor mínimo por columna
# 1    1
# 2    6
# 3    3
print(dataframe.min())

# Valor mínimo por fila
# a    1
# b    5
print(dataframe.min(axis=1))

# Valor máximo por columna
# 1    5
# 2    8
# 3    7
print(dataframe.max())

# Valor máximo por fila
# a    8
# b    7
print(dataframe.max(axis=1))
```

```
# Valor mínimo, pero indicando el índice
# 1    a
# 2    b
# 3    a
print(dataframe.idxmin())

# Valor máximo, pero indicando el índice
# 1    b
# 2    a
# 3    b
print(dataframe.idxmax())

# Estadistica
#              1         2         3
# count  2.000000  2.000000  2.000000 (Número de elementos)
# mean   3.000000  7.000000  5.000000 (Media)
# std    2.828427  1.414214  2.828427 (Desviación)
# min    1.000000  6.000000  3.000000 (Mínimo)
# 25%    2.000000  6.500000  4.000000 (25% del total)
# 50%    3.000000  7.000000  5.000000 (50% del total)
# 75%    4.000000  7.500000  6.000000 (75% del total)
# max    5.000000  8.000000  7.000000 (Máximo)
print(dataframe.describe())
```

- Valores nulos:

```
import pandas as pd
import numpy as np

# Para agregar un valor nulo se usa pn.nan
# ['1', '2', nan, '4']
lista_valores = ['1', '2', np.nan, '4']
print(lista_valores)

# a      1
# b      2
# c    NaN
# d      4
serie = pd.Series(lista_valores, index=list('abcd'))
print(serie)

# Verificamos los null
# a    False
# b    False
# c     True
# d    False
print(serie.isnull())
```

```python
# Eliminar los valores null
# a    1
# b    2
# d    4
print(serie.dropna())

# Borrarlos definitivos
# a    1
# b    2
# d    4
serie = serie.dropna()
print(serie)

# Generamos un dataframe
#    a    b    c
# 1  1  2.0  3.0
# 2  4  NaN  5.0
# 3  6  7.0  NaN
lista_valores = [[1, 2, 3], [4, np.nan, 5], [6, 7, np.nan]]
lista_indices = list('123')
lista_columnas = list('abc')
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)
```

```python
# Verificamos los nulos
#       a      b      c
# 1  False  False  False
# 2  False   True  False
# 3  False  False   True
print(dataframe.isnull())

# Eliminar filas con datos nulos
#   a    b    c
# 1  1  2.0  3.0
print(dataframe.dropna())

# Para poder operar sobre el dataframe, rellenamos los nulos con ceros
#    a    b    c
# 1  1  2.0  3.0
# 2  4  0.0  5.0
# 3  6  7.0  0.0
print(dataframe.fillna(0))
```

```python
# Mantener los cambios
#    a    b    c
# 1  1  2.0  3.0
# 2  4  0.0  5.0
# 3  6  7.0  0.0
dataframe = dataframe.fillna(0)
print(dataframe)
```

- Jerarquía de índices:

```python
import pandas as pd
import numpy as np

# Ej. [0.10364832 0.5824513  0.64426038 0.41962231 0.46428526 0.95431195]
lista_valores = np.random.rand(6)
lista_indices = [[1, 1, 1, 2, 2, 2], ['a', 'b', 'c', 'a', 'b', 'c']]

# Ej. Generamos la serie con dos indices
# 1  a    0.594231
#    a    0.020913
#    a    0.846184
# 2  b    0.130309
#    b    0.569375
#    b    0.683207
series = pd.Series(lista_valores, index=lista_indices)
print(series)
print(series[1])
print(series[1]['b'])

# Creamos un dataframe a partir de una serie con 2 indices
#          a         b         c
# 1  0.894156  0.086921  0.577181
# 2  0.702503  0.494718  0.908787
dataframe = series.unstack()
print(dataframe)

# Creamos un dataframe
#     a   b   c   d
# 1   0   1   2   3
# 2   4   5   6   7
# 3   8   9  10  11
# 4  12  13  14  15
lista_valores = np.arange(16).reshape(4,4)
lista_indices = list('1234')
lista_columnas = list('abcd')
dataframe2 = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)
```

```python
# Pasamos el dataframe a una Serie con doble indice
# 1  a     0
#    b     1
#    c     2
#    d     3
# 2  a     4
#    b     5
#    c     6
#    d     7
# 3  a     8
#    b     9
#    c    10
#    d    11
# 4  a    12
#    b    13
#    c    14
#    d    15
serie2 = dataframe2.stack()
print(serie2)
```

- Ejercicio 1:

```python
# - Hay 3 clases "clase1", "clase2", "clase3".
# - Generar un número aleatorio de alumnos por clase.
#   Se va a utilizar np.random.randint(minimo, maximo, numero)
# - Crear serie de clases y alumnos
# - Utilizar el índice de la serie para saber el número de alumnnos de la clase2

import pandas as pd
import numpy as np

minimo = 10
maximo = 40
numero_aleatorio = 3 # Ya que son 3 clases
alumnos = np.random.randint(minimo, maximo, numero_aleatorio)
# Ejemplo, [34 24 28]
print(alumnos)

clases = ['Clase_1', 'Clase_2', 'Clase_3']

# Clase_1    22
# Clase_2    34
# Clase_3    14
serie = pd.Series(alumnos, index=clases)
print(serie)

print(serie['Clase_2'])
```

- Ejercicio 2:

```python
# Dado el fichero excel en varios formatos.
# Copiar los datos al portapapeles.
import pandas as pd
# Crear un dataframe "datos" con esos datos copiados.
datos = pd.read_clipboard()
# Mostrar por pantalla, todos los datos del dataframe.
print(datos)
# Mostrar por pantalla, todos los nombres de columnas del dataframe.
print(datos.columns)
# Mostrar por pantalla, la primera fila del dataframe.
print(datos.loc[0])
# Mostrar por pantalla, la primera columna del dataframe.
print(datos['Continente'])
# Mostrar por pantalla, todas las filas pero solo las columnas "Continente" y "Población".
print(datos[['Continente', 'Población']])
# Mostrar por pantalla, las primeras 3 filas del dataframe.
print(datos.head(3))
# Mostrar por pantalla, las 2 últimas filas del dataframe.
print(datos.tail(2))
```

21. HTML

- HTML:

```python
import pandas as pd

url = 'https://es.wikipedia.org/wiki/Anexo:Finales_de_la_Copa_Mundial_de_F%C3%BAtbol'
dataframe = pd.io.html.read_html(url)
print(dataframe)

# Copiamos el dataframe
dataframe_futbol = dataframe[0]
print(dataframe_futbol)

# Mostramos una sola fila
print(dataframe_futbol.loc[0])

# Pasamos una fila a diccionario
print(dict(dataframe_futbol.loc[0]))

# Modificamos las columnas por los del diccionario.
diccionario = {'0':'Date', '1':'Champion', '2':'Result', '3':'Runner-Up', '4':'Stadium', '5':'Viewers', '6':'Location', '7':'Note'}
print(diccionario)
dataframe_futbol = dataframe_futbol.rename(columns=diccionario)
print(dataframe_futbol)

# Eliminamos una fila
dataframe_futbol = dataframe_futbol.drop(0)

 # Eliminamos una columna
dataframe_futbol = dataframe_futbol.drop('Notas', axis=1)
print(dataframe_futbol)
```

- Excel:

```
import pandas as pd
ficheroExcel = pd.ExcelFile('Libro1.xlsx')
misDatosExcel = ficheroExcel.parse('Hoja1')
print(misDatosExcel)
```

```
# Obtener la tabla de paises de la página https://es.wikipedia.org/wiki/Anexo:Pa%C3%ADses
# Limpiar los datos lo necesario para que los nombres de las columnas sean correctos.

import pandas as pd

excel = pd.ExcelFile('poblacion.xlsx')
dataframe = excel.parse('Hoja 1')
print(dataframe)
print(dataframe['Ciudad más poblada'][3])

csv = pd.read_csv('poblacion.csv')
print(csv)
print(csv['Ciudad más poblada'][1])
```

22. Tratamiento de datos con DataFrame

- Unión de dataframes:

```python
import pandas as pd

dataframe1 = pd.DataFrame({'c1':['1', '2', '3'], 'clave':['a', 'b', 'c']})
print(dataframe1)

dataframe2 = pd.DataFrame({'c2':['4', '5', '6'], 'clave':['c', 'b', 'e']})
print(dataframe2)

# Unimos los 2 dataframe
dataframe3 = pd.DataFrame.merge(dataframe1, dataframe2)
print(dataframe3)

# Unimos los 2 dataframe indicando la clave
dataframe3 = pd.DataFrame.merge(dataframe1, dataframe2, on='clave')
print(dataframe3)

# Unimos los 2 dataframe indicando la clave y que mantenga el izquierdo, derecho y ambos
dataframe4 = pd.DataFrame.merge(dataframe1, dataframe2, on='clave', how='left')
print(dataframe4)
dataframe4 = pd.DataFrame.merge(dataframe1, dataframe2, on='clave', how='right')
print(dataframe4)
dataframe4 = pd.DataFrame.merge(dataframe1, dataframe2, on='clave', how='outer')
print(dataframe4)
```

- Concatenación de datos:

```
import pandas as pd
import numpy as np

# Generamos 1 array
array1 = np.arange(9).reshape(3,3)
print(array1)
# Concatenamos el array
array_concatenado = np.concatenate([array1, array1])
print(array_concatenado)
# Concatenamos el array por columna
array_concatenado2 = np.concatenate([array1, array1], axis=1)
print(array_concatenado2)

# Generamos 2 series
serie1 = pd.Series([1,2,3], index=['a', 'b', 'c'])
serie2 = pd.Series([4,5,6], index=['d', 'e', 'f'])
# Concatenamos la serie
serie_concatenada = pd.concat([serie1, serie2])
print(serie_concatenada)
# concatenamos colocandole una llave extra
serie_concatenadaLlave = pd.concat([serie1, serie2], keys=['Serie_1', 'Serie_2'])
print(serie_concatenadaLlave)
```

```
# Generamos 2 dataframe
dataframe1 = pd.DataFrame(np.random.rand(3,3), columns=['a', 'b', 'c'])
dataframe2 = pd.DataFrame(np.random.rand(2,3), columns=['a', 'b', 'c'])
print(dataframe1)
print(dataframe2)
# Concatenamos los dataframe
dataframeConcatenado = pd.concat([dataframe1, dataframe2])
print(dataframeConcatenado)
# Reorganizamos el índice para que se muestre correctamente
dataframeConcatenadoReorg = pd.concat([dataframe1, dataframe2], ignore_index=True)
print(dataframeConcatenadoReorg)
```

- Combinar Series y Dataframes:

```
import pandas as pd
import numpy as np

serie1 = pd.Series([1,2,np.nan])
serie2 = pd.Series([4,5,6])
# Combinamos
serie3 = serie1.combine_first(serie2)
print(serie3)

lista_valores = [1,2,np.nan]
dataframe1 = pd.DataFrame(lista_valores)
print(dataframe1)
lista_valores2 = [4,5,6]
dataframe2 = pd.DataFrame(lista_valores2)
print(dataframe2)

dataframe3 = dataframe1.combine_first(dataframe2)
print(dataframe3)
```

- Duplicado Dataframe:

```
import pandas as pd

lista_valores = [[1,2], [1,2], [5,6], [5,8]]
lista_indices = list('nmop')
lista_columnas = ['valor1', 'valor2']
print(lista_valores)
print(lista_indices)
print(lista_columnas)
dataframe = pd.DataFrame(lista_valores, index=lista_indices, columns=lista_columnas)
print(dataframe)

# Eliminamos los duplicados
print(dataframe.drop_duplicates())
# Eliminamos duplicados dentro de una columna
print(dataframe.drop_duplicates(['valor1']))
```

- Reemplazar datos en serie:

```
import pandas as pd

serie = pd.Series([1,2,3,4,5], index=list('abcde'))
print(serie)

# reeplazamos el 1 por el 6
print(serie.replace(1,6))

# reemplazamos por una serie
print(serie.replace({2:8, 3:9}))
```

- Renombrar índices:

```
import pandas as pd
import numpy as np

lista_valores  = np.arange(9).reshape(3,3)
print(lista_valores)
lista_indices = list('abc')
print(lista_indices)

dataframe = pd.DataFrame(lista_valores, index=lista_indices)
print(dataframe)

# Pasamos el indice a mayuscula
nuevos_indices = dataframe.index.map(str.upper)
dataframe.index = nuevos_indices
print(dataframe)

# Otra forma es
dataframe = dataframe.rename(index=str.lower)
print(dataframe)

# Cambiamos el índice
nuevos_indices = {'a':'f', 'b':'g', 'c':'h'}
print(dataframe.rename(index=nuevos_indices))
```

- Agrupar datos en categorías:

```python
import pandas as pd
import numpy as np

precios = [42, 55, 48, 23, 5, 21, 88, 34, 26]
rango = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# [(40, 50], (50, 60], (40, 50], (20, 30], (0, 10], (20, 30],
# (80, 90], (30, 40], (20, 30]]
# Categories (10, interval[int64]): [(0, 10] < (10, 20] <
# (20, 30] < (30, 40] ... (60, 70] < (70, 80] < (80, 90] <
# (90, 100]]
precios_con_rango = pd.cut(precios, rango)
print(precios_con_rango)

# Contamos cuantos precios hay en cada categoria
# (20, 30]     3
# (40, 50]     2
# (0, 10]      1
# (30, 40]     1
# (50, 60]     1
# (80, 90]     1
# (10, 20]     0
# (60, 70]     0
# (70, 80]     0
# (90, 100]    0
# dtype: int64
print(pd.value_counts(precios_con_rango))
```

- Filtrar datos en DataFrames:

```python
import pandas as pd
import numpy as np

# Generamos una lista de 10 filas x 3 columnas
lista_valores = np.random.rand(10,3)
print(lista_valores)

# DataFrame
dataframe = pd.DataFrame(lista_valores)
print(dataframe)

# Columna 0
columna = dataframe[0]
print(columna)
print(columna[columna > 0.40])
```

- Combinación de elementos:

```python
import pandas as pd
import numpy as np

lista_valores = np.arange(25).reshape(5,5)
print(lista_valores)

dataframe = pd.DataFrame(lista_valores)
print(dataframe)

combinacion_aleatoria = np.random.permutation(5)
print(combinacion_aleatoria)
print(dataframe.take(combinacion_aleatoria))
```

- Agrupación de Dataframe:

```
import pandas as pd
import numpy as np

lista_valores = {'clave1':['x', 'x', 'y', 'y', 'z'], 'clave2':['a', 'b', 'a', 'b', 'a'],
'datos1':np.random.rand(5), 'datos2':np.random.rand(5)}
print(lista_valores)

# Generamos el DataFrame
dataframe = pd.DataFrame(lista_valores)
print(dataframe)

# Ej. agrupamos datos1 a través de la clave1
grupo1 = dataframe['datos1'].groupby(dataframe['clave1'])
print(grupo1)
print(grupo1.mean())
```

- Agregación DataFrame:

```
import pandas as pd
import numpy as np

lista_valores = ([1,2,3], [4,5,6], [7,8,9], [np.nan, np.nan, np.nan])
lista_columnas = list('abc')
dataframe = pd.DataFrame(lista_valores, columns=lista_columnas)
print(dataframe)

# Operaciones por columnas
#         a     b     c
# sum  12.0  15.0  18.0
# min   1.0   2.0   3.0
# max   7.0   8.0   9.0
print(dataframe.agg(['sum', 'min', 'max']))

# Operaciones por fila
# 0     6.0
# 1    15.0
# 2    24.0
# 3     0.0
print(dataframe.agg('sum', axis=1))
```

- Ejemplos 1:

```python
import pandas as pd
import numpy as np
# - Crearemos una lista de 50 valores aleatorios enteros entre los
# valores 0 y 100.
lista_valores = np.random.randint(0,100,50)
print(lista_valores)
# - Convertiremos esta lista en un dataframe con 5 filas y 10 columnas
lista_valores.resize(5,10)
dataframe = pd.DataFrame(lista_valores)
print(dataframe)
# - Filtraremos los datos del dataframe para visualizar solo los valores
# que sean mayores de 50.
print(dataframe[dataframe > 50])
```

- Ejemplo 2:

```python
import pandas as pd
import numpy as np

# Crear 2 array con 9 números aleatorios enteros entre los valores 0 y 100
array1 = np.random.randint(0,100,9)
array2 = np.random.randint(0,100,9)
# Cambiar el formato de los arrays en una estructura de 3 filas x 3 columnas
array1 = array1.reshape(3,3)
array2 = array2.reshape(3,3)
# Crear 2 fataframe con esos arrays
dataframe1 = pd.DataFrame(array1)
dataframe2 = pd.DataFrame(array2)
# Concatenar esos 2 dataframes
dataframe_concatenado = pd.concat([dataframe1, dataframe2], ignore_index=True)
print(dataframe_concatenado)
```

23. Módulo seaborn y matplotlib

- Sirve para hacer gráficos estadísticos.
- Instalación:

```
>pip install seaborn
```

- Historigramas: Representación gráfica de una variable o unos datos en forma de barras. Donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

```
import pandas as pd
import numpy as np
```

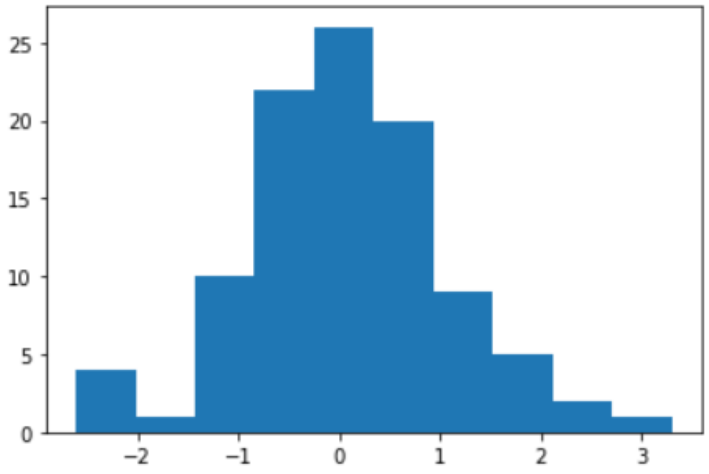
```
import matplotlib as mpl
import seaborn as sns
```

```
# Para que muestre el gráfico
%matplotlib inline
```

```
datos1 = np.random.randn(100)
```

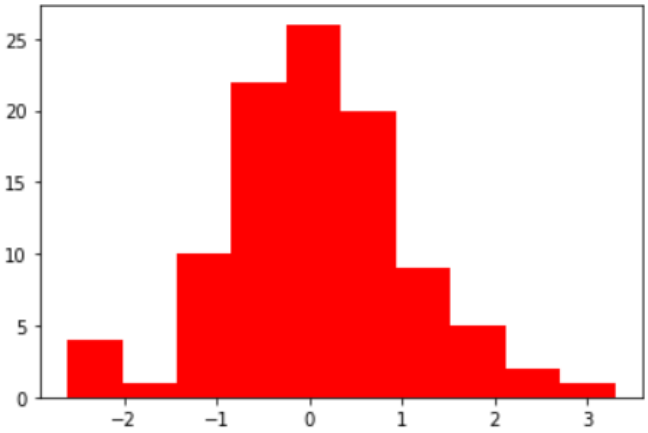
```
mpl.pyplot.hist(datos1)
```

```
(array([ 4.,  1., 10., 22., 26., 20.,  9.,  5.,  2.,  1.]),
 array([-2.61238211, -2.02127901, -1.4301759 , -0.83907279, -0.24796969,
         0.34313342,  0.93423652,  1.52533963,  2.11644274,  2.70754584,
         3.29864895]),
 <a list of 10 Patch objects>)
```



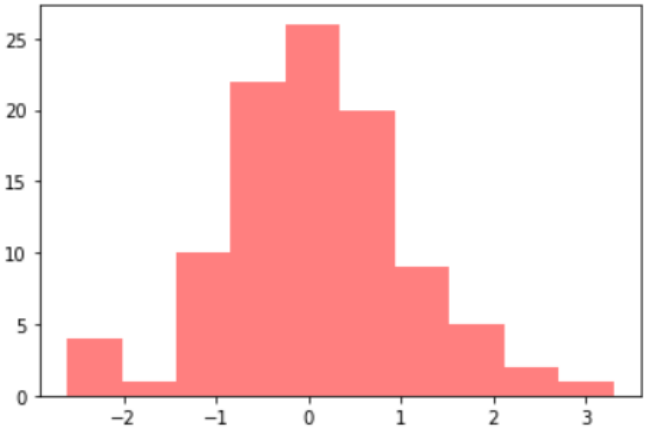
```
mpl.pyplot.hist(datos1, color="red")
```

```
(array([ 4.,  1., 10., 22., 26., 20.,  9.,  5.,  2.,  1.]),
 array([-2.61238211, -2.02127901, -1.4301759 , -0.83907279, -0.24796969,
         0.34313342,  0.93423652,  1.52533963,  2.11644274,  2.70754584,
         3.29864895]),
 <a list of 10 Patch objects>)
```




```
mpl.pyplot.hist(datos1, color="red", alpha=0.5)
```

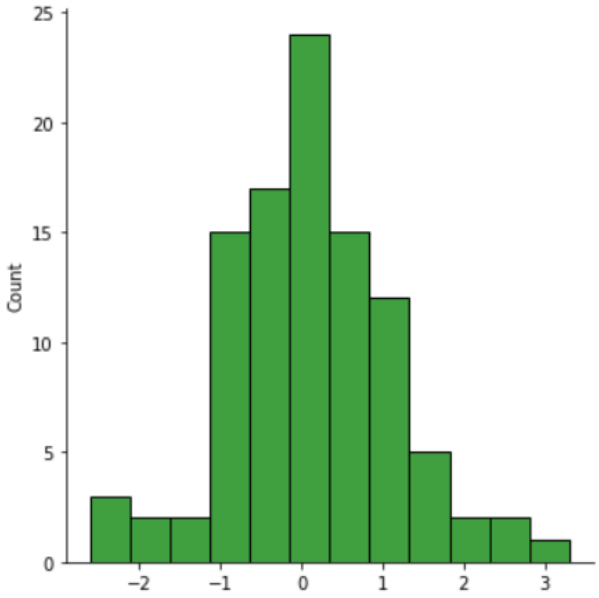
```
(array([ 4.,  1., 10., 22., 26., 20.,  9.,  5.,  2.,  1.]),
 array([-2.61238211, -2.02127901, -1.4301759 , -0.83907279, -0.24796969,
         0.34313342,  0.93423652,  1.52533963,  2.11644274,  2.70754584,
         3.29864895]),
 <a list of 10 Patch objects>)
```



Ahora también se puede utilizar el módulo gráfico sns.

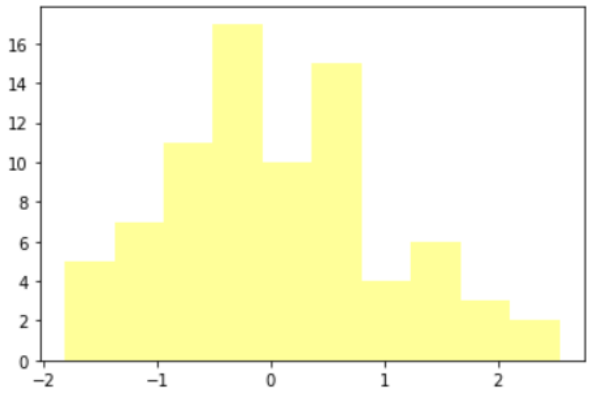
```
sns.displot(datos1, color="green")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb800454ad0>
```




```
mpl.pyplot.hist(datos2, color="yellow", alpha=0.4)
```

```
(array([ 5.,  7., 11., 17., 10., 15.,  4.,  6.,  3.,  2.]),
 array([-1.80956361, -1.37464895, -0.9397343 , -0.50481964, -0.06990498,
         0.36500968,  0.79992433,  1.23483899,  1.66975365,  2.10466831,
         2.53958296]),
 <a list of 10 Patch objects>)
```



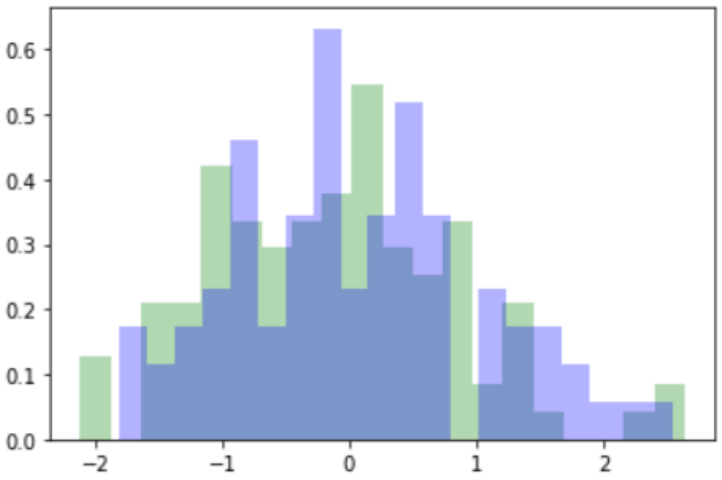
Generamos 2 gráficos:

```
datos1 = np.random.randn(100)
```

```
datos2 = np.random.randn(80)
```

```
mpl.pyplot.hist(datos1, color="green", alpha=0.3, bins=20, density=True)
mpl.pyplot.hist(datos2, color="blue", alpha=0.3, bins=20, density=True)
```

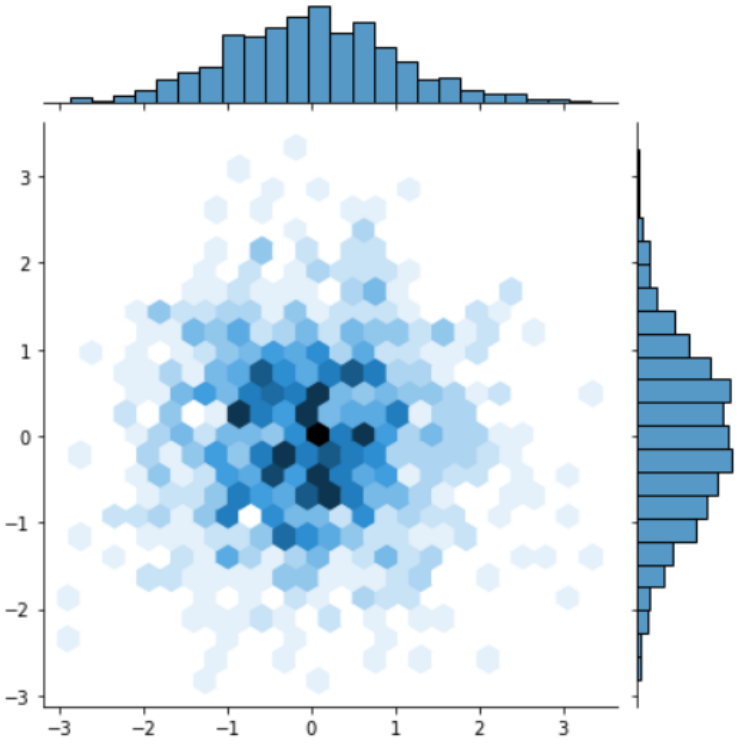
```
(array([0.17244763, 0.11496508, 0.17244763, 0.22993017, 0.45986033,
        0.17244763, 0.34489525, 0.63230796, 0.22993017, 0.34489525,
        0.51734288, 0.34489525, 0.        , 0.22993017, 0.17244763,
        0.17244763, 0.11496508, 0.05748254, 0.05748254, 0.05748254]),
 array([-1.80956361, -1.59210628, -1.37464895, -1.15719162, -0.9397343 ,
        -0.72227697, -0.50481964, -0.28736231, -0.06990498,  0.14755235,
         0.36500968,  0.58246701,  0.79992433,  1.01738166,  1.23483899,
         1.45229632,  1.66975365,  1.88721098,  2.10466831,  2.32212564,
         2.53958296]),
 <a list of 20 Patch objects>)
```



```
sns.jointplot(datos3, datos4, kind="hex")
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seabor
version 0.12, the only valid positional argument will be `d
interpretation.
  FutureWarning
```

```
<seaborn.axisgrid.JointGrid at 0x7f2e57738f10>
```



- Distribuciones:

    - Combinar estilos:

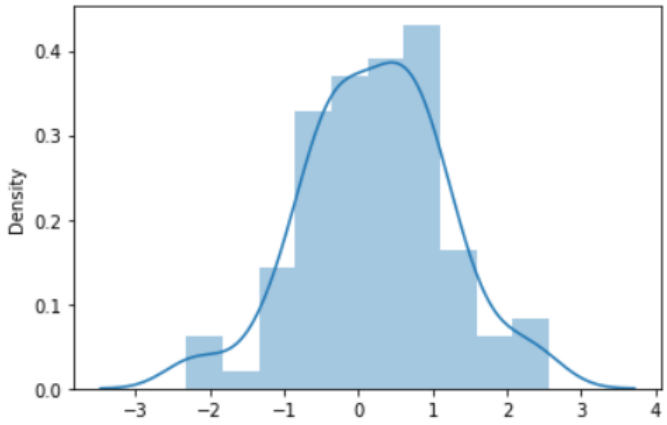
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import seaborn as sns
%matplotlib inline
```

```
datos = np.random.randn(100)
```

```
sns.distplot(datos)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seabo
ved in a future version. Please adapt your code to use eit
function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0c8a56190>
```

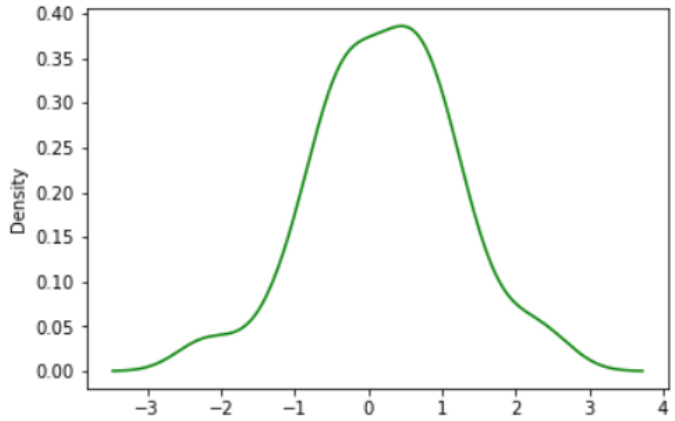


Eliminamos las barras:

```
sns.distplot(datos, color="green", rug=False, hist=False)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seabo
ved in a future version. Please adapt your code to use eit
function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0bfbde850>
```

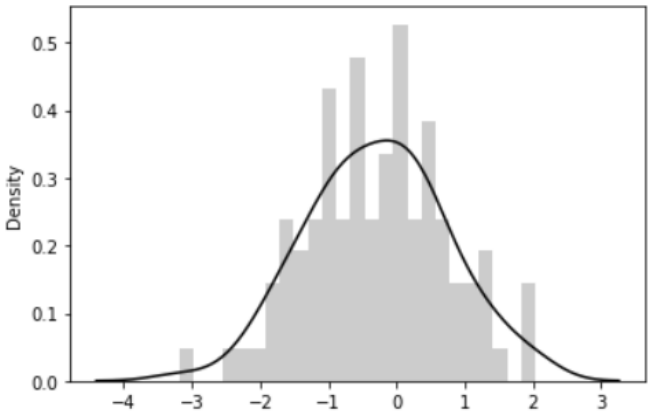


Color a la curva y color a las barras:

```
argumentos_curva = {'color':'black', 'label':'Curva'}
argumentos_histograma = {'color':'grey', 'label':'Histograma'}
sns.distplot(datos, bins=25, kde_kws=argumentos_curva, hist_kws=argumentos_histograma)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/distributions.py:2557: Fut
ved in a future version. Please adapt your code to use either `displot` (a figure-level
function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5968025fd0>
```



- Gráficos de caja:

    - Sirve para representar gráficamente una serie de datos numéricos a través de sus cuartiles.

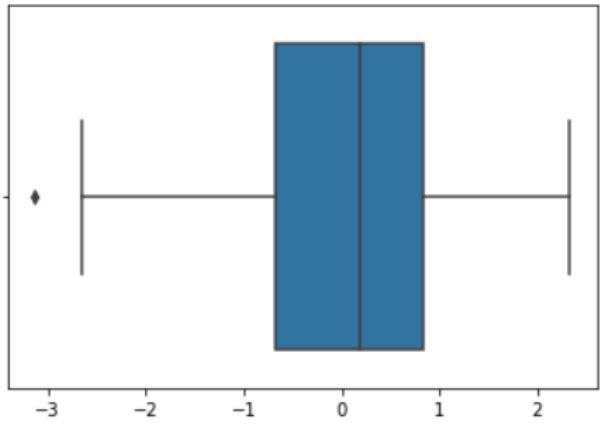
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import seaborn as sns
%matplotlib inline
```

```
datos = np.random.randn(100)
```

```
sns.boxplot(datos)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seabo
rsion 0.12, the only valid positional argument will be `da
terpretation.
  FutureWarning
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01a58113d0>
```



- Regresiones lineales: proceso estadístico para estimar las relaciones entre variables, es decir, entender como varia el valor de una variable en función del valor de otras variables.

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

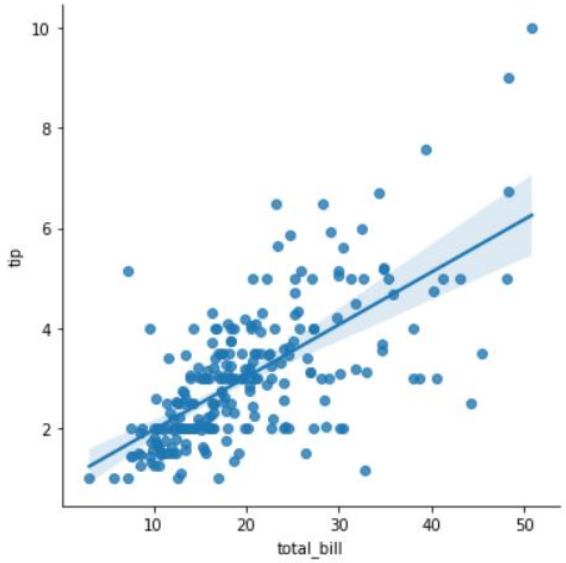
```
propinas = sns.load_dataset('tips')
propinas.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
sns.lmplot('total_bill', 'tip', propinas)
```

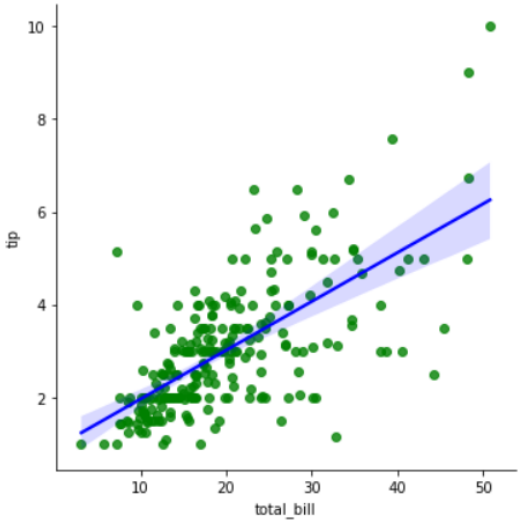
```
/srv/conda/envs/notebook/lib/python3.7/site-packages/
a. From version 0.12, the only valid positional argum
r or misinterpretation.
  FutureWarning
```

```
<seaborn.axisgrid.FacetGrid at 0x7f67bc1c8e50>
```



```
sns.lmplot('total_bill','tip',propinas,scatter_kws={'marker':'o','color':'green'},line_kws={'color':'blue'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f678e4a02d0>
```

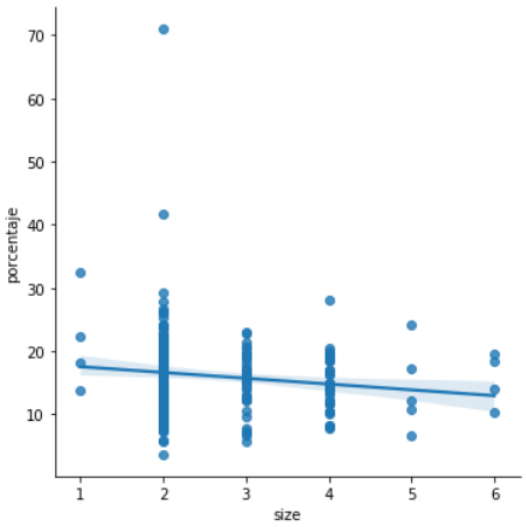



```
propinas['porcentaje'] = 100 * propinas['tip'] / propinas['total_bill']
propinas.head()
```

	total_bill	tip	sex	smoker	day	time	size	porcentaje
0	16.99	1.01	Female	No	Sun	Dinner	2	5.944673
1	10.34	1.66	Male	No	Sun	Dinner	3	16.054159
2	21.01	3.50	Male	No	Sun	Dinner	3	16.658734
3	23.68	3.31	Male	No	Sun	Dinner	2	13.978041
4	24.59	3.61	Female	No	Sun	Dinner	4	14.680765

```
sns.lmplot('size','porcentaje',propinas)
```

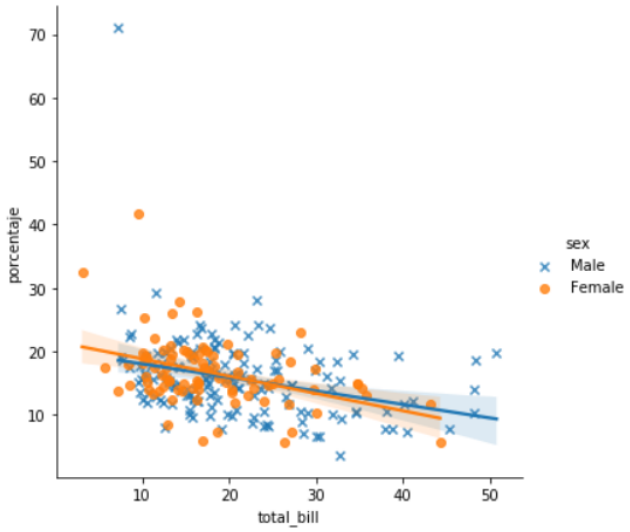
```
<seaborn.axisgrid.FacetGrid at 0x7f678e412690>
```



```
sns.lmplot('total_bill','porcentaje',propinas,hue='sex',markers=['x','o'])
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorators.py
a. From version 0.12, the only valid positional argument will be `data`, and
r or misinterpretation.
  FutureWarning
```

```
<seaborn.axisgrid.FacetGrid at 0x7f678e28db50>
```



- Mapas de calor: Un mapa de calor es una representación gráfica de datos, donde los valores individuales contenidos en una matriz se representan como colores.

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

```
# Seaborn tiene un conjunto de datos dataset
# precargados que se pueden utilizar para pruebas
# en este caso uno de vuelos
vuelos = sns.load_dataset('flights')
vuelos
```

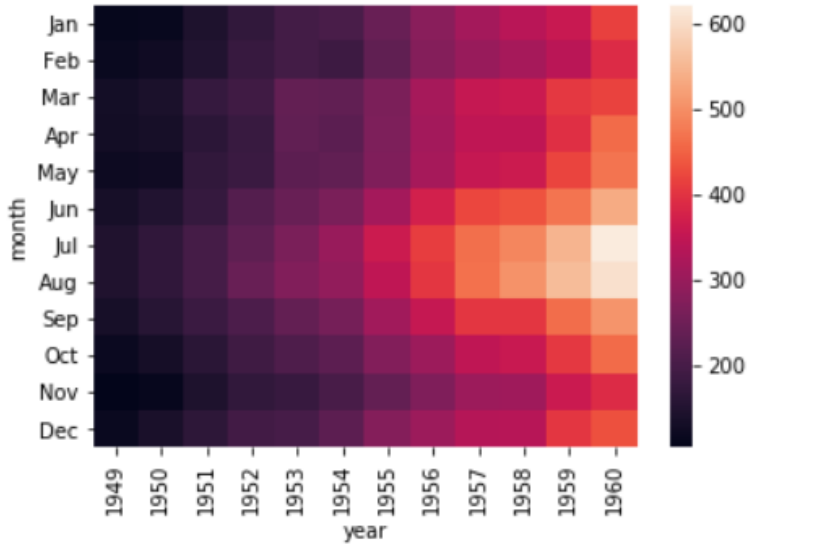
	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121
...	...	...	...
139	1960	Aug	606

```
# Se genera una matriz de datos
vuelos = vuelos.pivot('month', 'year', 'passengers')
vuelos
```

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
Jan	112	115	145	171	196	204	242	284	315	340	360	417
Feb	118	126	150	180	196	188	233	277	301	318	342	391
Mar	132	141	178	193	236	235	267	317	356	362	406	419
Apr	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
Jun	135	149	178	218	243	264	315	374	422	435	472	535
Jul	148	170	199	230	264	302	364	413	465	491	548	622

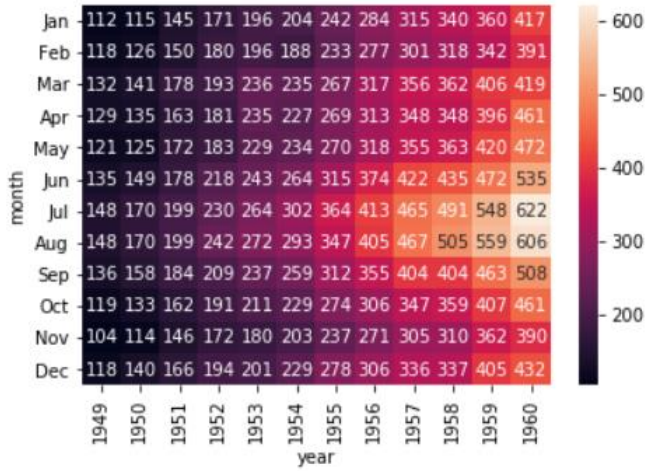
```
# Generamos el mapa de calos
sns.heatmap(vuelos)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8efde70d0>
```



```
# Mostrando lo valores
sns.heatmap(vuelos, annot=True, fmt='d')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8efc9eb90>
```

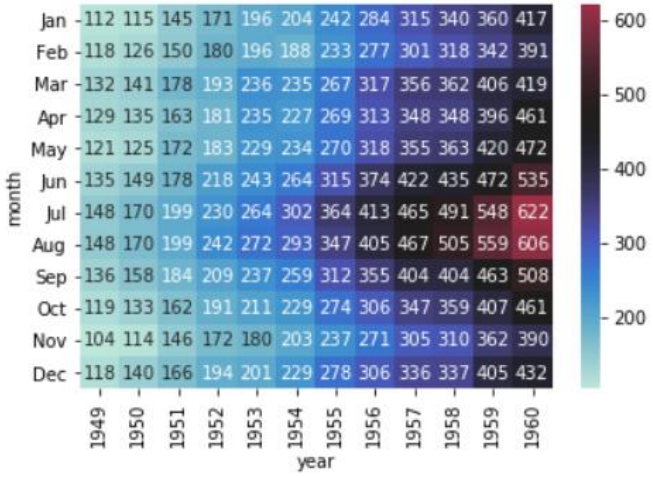



```
# Obtenemos un valor del mapa
vuelos.loc['Aug'][1957]
```

```
467
```

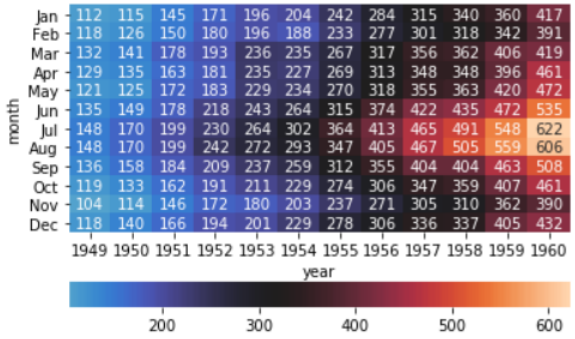
```
# Obtenemos un valor del mapa
valor_central = vuelos.loc['Aug'][1957]
sns.heatmap(vuelos, center=valor_central, annot=True, fmt='d')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8efbe2410>
```



```
# Se cambia la barra verticar a la derecho y se pondrá abajo horizontal
sns.heatmap(vuelos, center=valor_central, annot=True, fmt='d', cbar_kws={'orientation':'horizontal'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8b007a910>
```



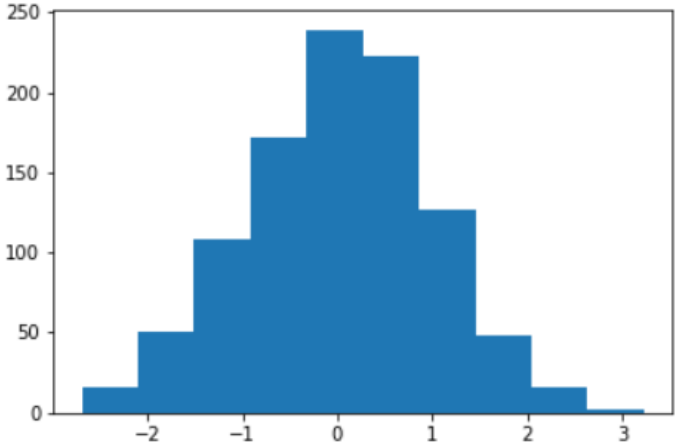
- Ejemplo 1:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# - Crear una lista de 1000 valores aleatorios que sigan
# una distribución normal.
lista_valores = np.random.randn(1000)
```

```
# - Crear un histograma mediante matplotlib con la lista
# de valores.
plt.hist(lista_valores)
```

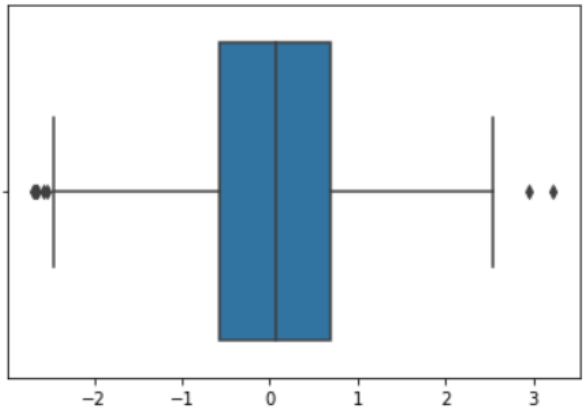
```
(array([ 16.,  50., 108., 172., 239., 222., 127.,  48.,  16.,   2.]),
 array([-2.69462522, -2.10229059, -1.50995596, -0.91762132, -0.32528669,
         0.26704794,  0.85938257,  1.4517172 ,  2.04405183,  2.63638646,
         3.22872109]),
 <a list of 10 Patch objects>)
```



```
# Crear un diagrama de caja (donde se acumula el 50% de los
# valores) mediante seaborn.
sns.boxplot(lista_valores)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/
rsion 0.12, the only valid positional argument will be `data`
terpretation.
  FutureWarning
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58d2d83690>
```



- Ejemplo 2:

```
import numpy as np
import seaborn as sns
```

```
# - Crear un array con 100 números enteros aleatorios con valores
# comprendidos entre 0 y 500
datos = np.random.randint(0,500,100)
```

```
# - utilizar un diagrama de caja para representar los valores
# aleatorios generados
sns.boxplot(datos)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decor
rsion 0.12, the only valid positional argument will be `data`, and
terpretation.
  FutureWarning
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc9c56d1150>
```