PYTHON
DIAGRAMA DE VENN
CON MATPLOTLIB

Un diagrama de Venn (también llamado diagrama primario, diagrama de conjuntos o diagrama lógico) es un diagrama que muestra todas las posibles relaciones lógicas entre una colección finita de conjuntos diferentes.

Cada conjunto está representado por un círculo. El tamaño del círculo a veces representa la importancia del grupo, pero no siempre. Los grupos suelen superponerse: el tamaño de la superposición representa la intersección entre ambos grupos.

Aquí hay un ejemplo que muestra el número de palabras compartidas en las letras de 3 cantantes franceses: Nekfeu , Booba y Georges Brassens : https://www.data-to-viz.com/story/venn.png

Un diagrama de Venn hace un buen trabajo para estudiar la intersección entre 2 o 3 conjuntos pero se vuelve muy difícil leer con más grupos .

Aquí hay un ejemplo de un diagrama de Venn de seis conjuntos publicado en Nature que muestra la relación entre el genoma del plátano y el genoma de otras cinco especies: https://www.nature.com/articles/nature11241/figures/4

Para aprender más sobre Diagramas de Venn ver:

https://en.wikipedia.org/wiki/Venn_diagram#Edwards.27_Venn_diagrams

Diagrama de Venn - Wikipedia, la enciclopedia libre

Propósitos y beneficios

- Organizar información visualmente para ver la relación entre los conjuntos de elementos, como semejanzas y diferencias.
- Comparar dos o más opciones y ver claramente lo que tienen en común y lo que puede distinguirlos.
- Resolver problemas matemáticos complejos .
- Comparar conjuntos de datos, encontrar correlaciones y predecir probabilidades de determinados acontecimientos.
- Razonar la lógica detrás de declaraciones o ecuaciones.

Usos en diferentes campos

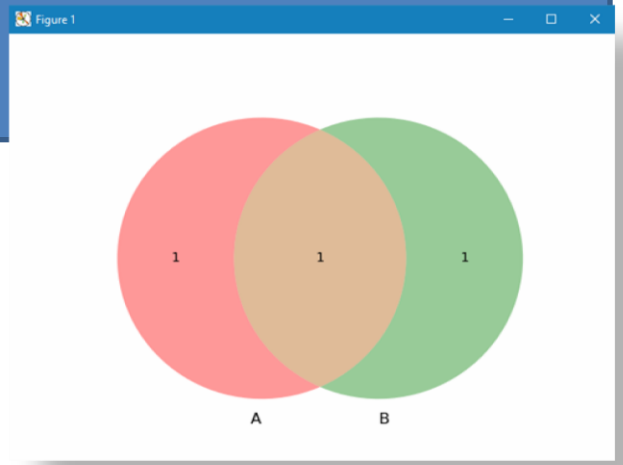
- Matemática: La teoría de conjuntos es una rama completa de la matemática.
- Estadística y probabilidad: se utilizan para predecir la probabilidad de determinados acontecimientos. Esto se relaciona con el campo del análisis predictivo.
- Lógica: se utilizan para determinar la validez de conclusiones y argumentos específicos.
- Lingüística: se utilizan para estudiar las diferencias y similitudes entre idiomas.
- Negocios: se utilizan para comparar y contrastar productos, servicios, procesos , etc. Son una herramienta de comunicación efectiva para ilustrar esa comparación.
- Psicología
- Ingeniería
- Planes de marketing
- Estudios de mercado
- Estudios demográficos

La biblioteca matplotlib-venn ha sido creada por Konstantin Tretyakov
Lo primero que tenemos que hacer es instalar la librería:
pip install matplotlib_venn
este comando también instalará las dependencias requeridas: SciPy, NumPy y matplotlib.

```
from matplotlib import pyplot as plt
# importamos  de matplotlib el módulo pyplot . La misma instrucción se podría haber expresado así:  import matplotlib.pyplot as plt.

from matplotlib_venn import venn2
# importamos de matplotlib_venn,  el módulo venn2,  que
# tiene implementado  clases, funciones, métodos, etc,
# para generar diagramas de Venn.

venn2((1, 1, 0))
plt.show()
```

Es un Diagrama de Venn?...


```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

venn2((1, 1, 1)) # generamos un diagrama  base de  2  conjuntos.
plt.show()
```

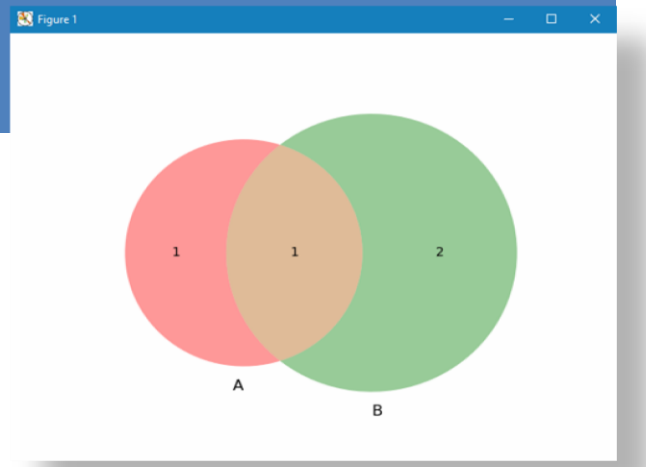


El argumento (1,1,1) indica el tamaño relativo de los tres subconjuntos en este
orden: Ab (izquierda), aB (derecha), AB (intersección).

- Ab = contenido en el grupo A, pero no B
- aB = Contenido en el grupo B, pero no A
- AB = contenido en los grupos A y B

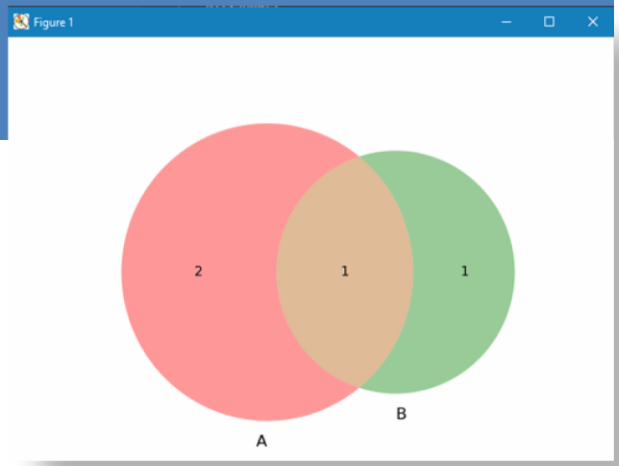
Así, la tupla (1, 2, 1) dibujaría el conjunto B del doble de tamaño respecto de A:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2
venn2((1, 2, 1))
plt.show()
```



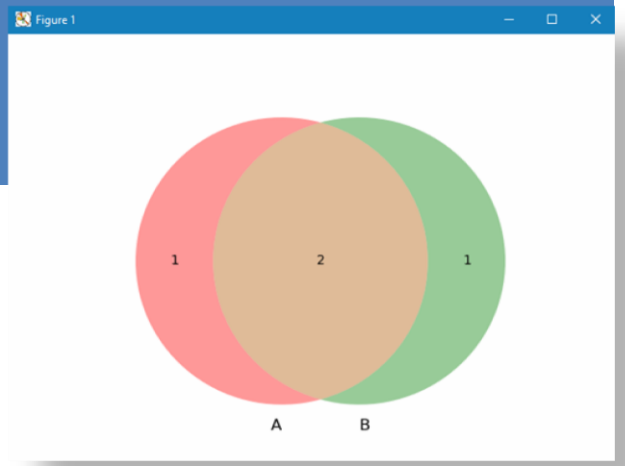
Y la tupla (2, 1, 1) dibujaría el conjunto A del doble de tamaño respecto de B:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2
venn2((2, 1, 1))
plt.show()
```



Y qué sucedería con la tupla (1, 1,2) …

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2
venn2((1, 1, 2))
plt.show()
```

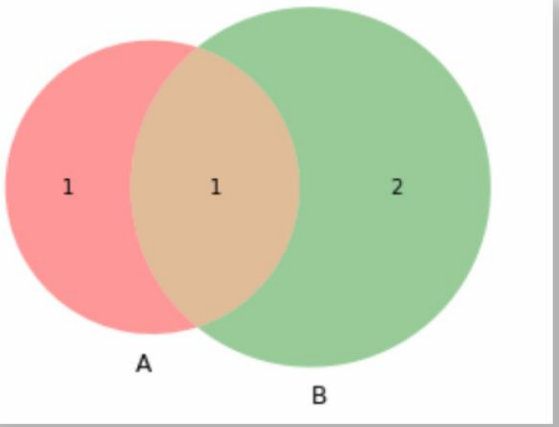


Para identificar a cada uno de los subconjuntos (3 en diagramas de 2 conjuntos) el módulo utiliza una nomenclatura que consiste en colocar un 1 para indicar que la sección está incluida en el conjunto y un 0 para indicar que está excluida.

De esta manera, siguiendo el orden «ABC», el subconjunto 10 (Ab) es el de la izquierda (el que pertenece a A pero no a B); el 01 (aB) , el de la derecha (el que pertenece a B pero no a A); y el 11 (AB), el del medio (la intersección).

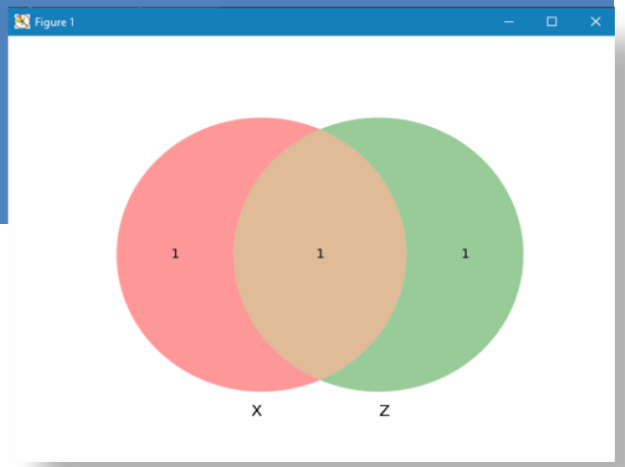
Teniendo esto en cuenta, como primer argumento , también podemos pasarle como parámetro un **diccionario** indicando el tamaño de cada uno de los subconjuntos:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2
venn2({"10": 1, "01": 2, "11": 1})
plt.show()
```



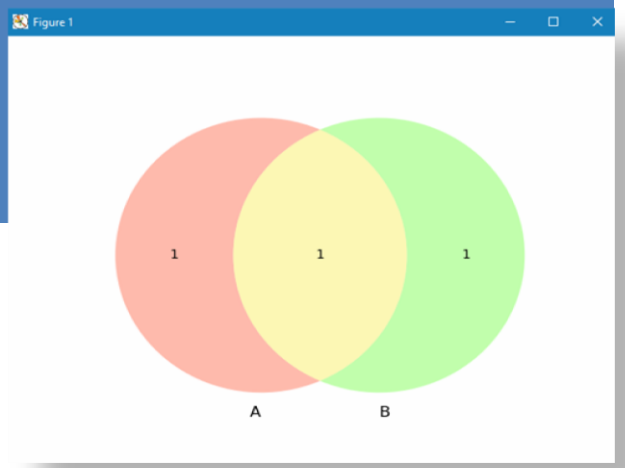
El parámetro **set_labels** permite cambiar los nombres de los conjuntos mostrados en el diagrama:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

venn2((1, 1, 1), set_labels=("X", "Z"))
plt.show()
```



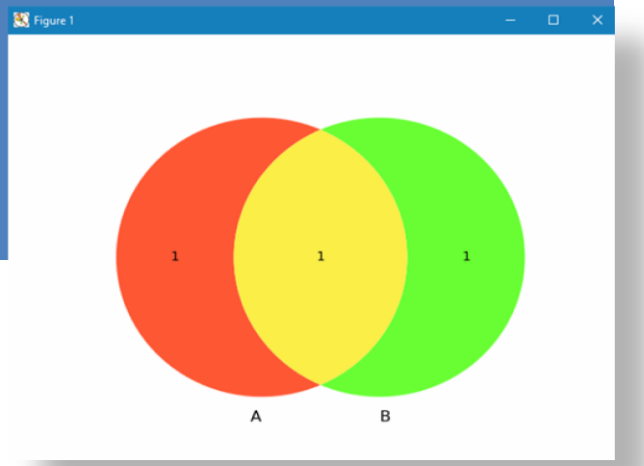
set_colors determina el color de los conjuntos. Nótese que por defecto tiene un alpha (transparencia) de 0.4:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

venn2((1, 1, 1), set_colors=("#FF5733", "#68FF33"))
plt.show()
```



Deshabilitamos la transparencia:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

venn2((1, 1, 1), set_colors=("#FF5733", "#68FF33"), alpha=1)
plt.show()
```



La función **venn2()** retorna una instancia de una clase llamada **VennDiagram**.

Los métodos de ésta clase que nos interesan son:

- **get_label_by_id()**
- **get_patch_by_id()**

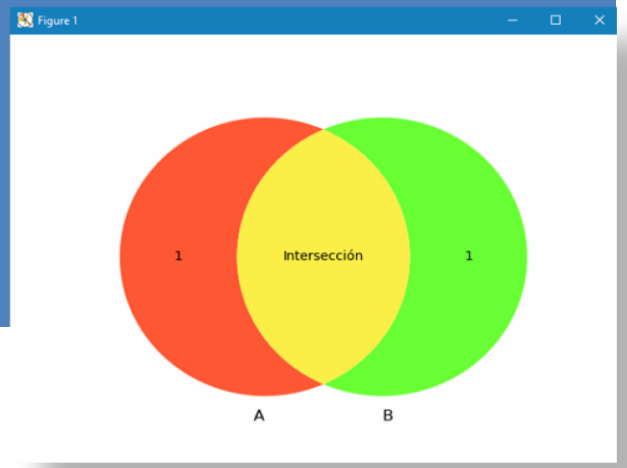
Ambas toman un subconjunto y retornan instancias de :

- **matplotlib.text.Text**
- **matplotlib.patches.PathPatch**

A partir de ellas podemos modificar individualmente el aspecto de cada subconjunto.

Por ejemplo, el siguiente código agrega la palabra “Intersección” al subconjunto 11 (AB) y establece su color:

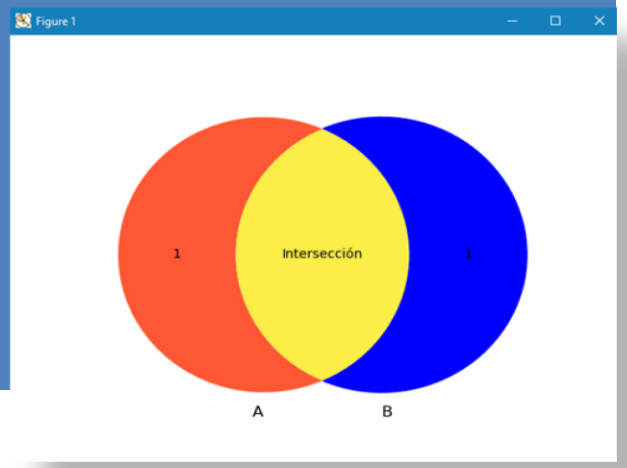
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"),  alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
plt.show()
```



Obsérvese que, previamente, asignamos a la variable “diagram”, el diagrama y es para escribir un código más claro y ordenado.

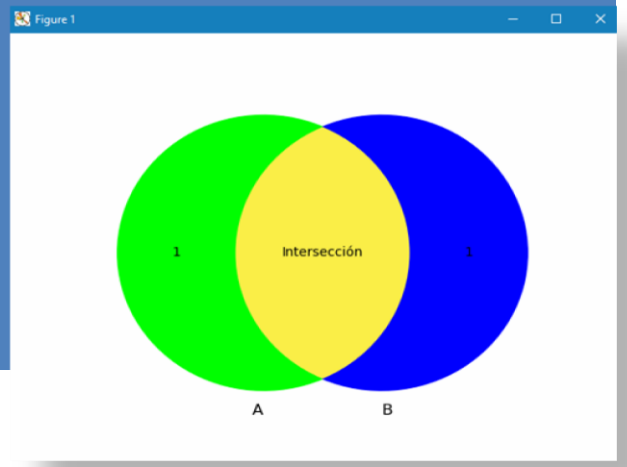
Cambiamos el color del conjunto B:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
plt.show()
```



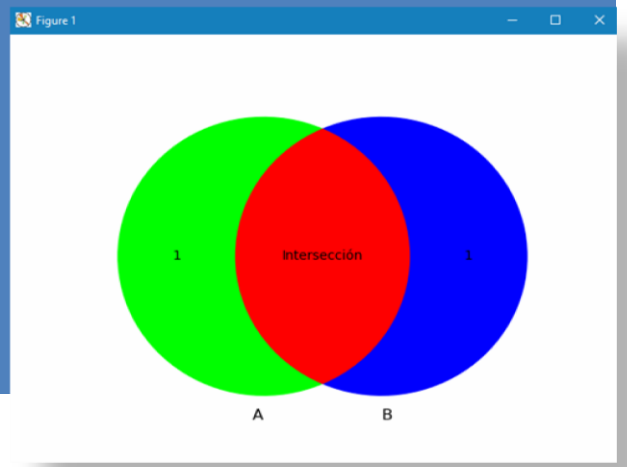
Cambiamos el color del conjunto A:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
plt.show()
```



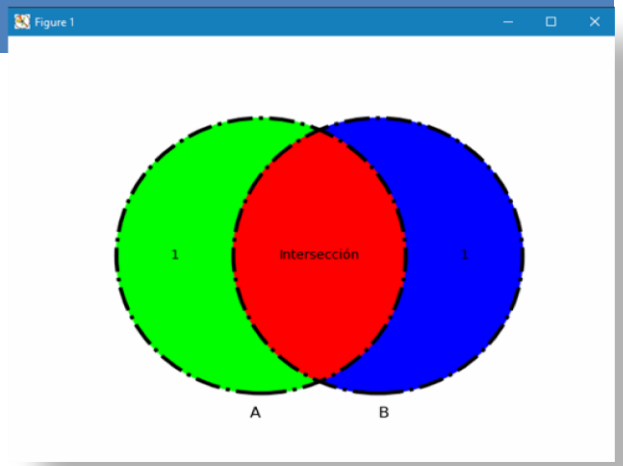
Cambiamos el color de la intersección:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")
plt.show()
```



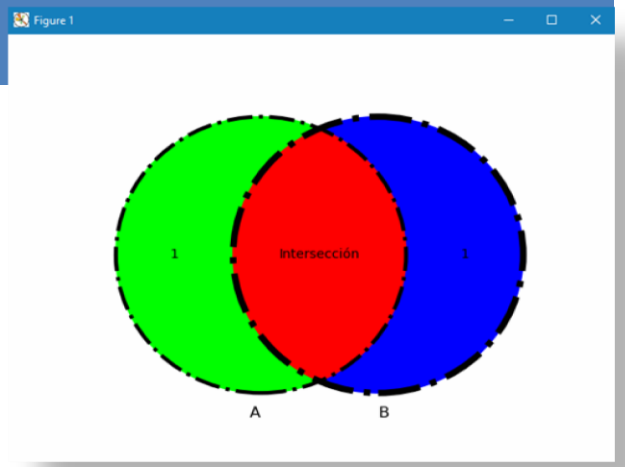
Le agregamos líneas con color al diagrama, para eso tenemos que importar el módulo **venn2_circles**:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

venn2_circles(subsets=(1, 1, 1), color="#000000", alpha=1, linestyle="-.", linewidth=3)
plt.show()
```



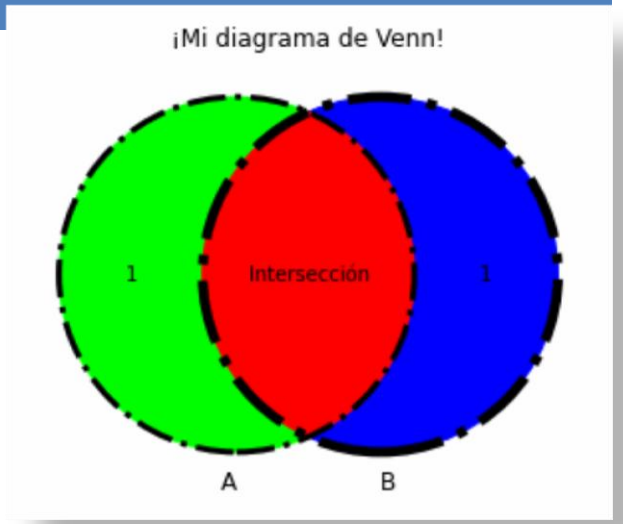
Si configuramos el círculo del diagrama de Venn en la variable ‘c’, puedo llamar a cada círculo individual c [0] y c [1] y establecer el ancho de línea:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

c = venn2_circles(subsets=(1, 1, 1), color="#000000", alpha=1, linestyle="-.", linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
plt.show()
```



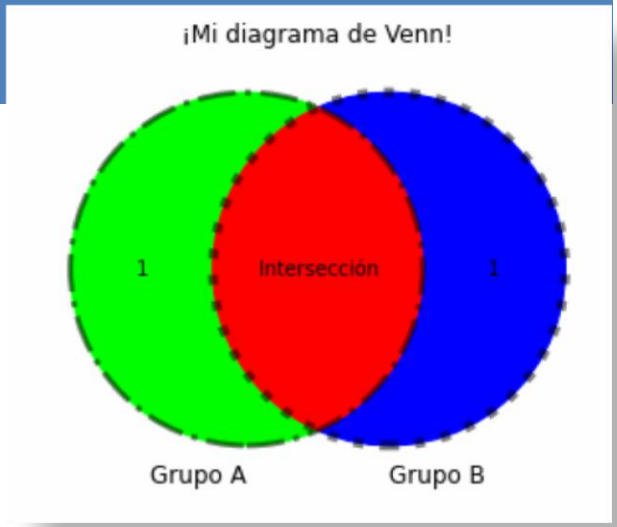
Y le colocamos un título:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF") .
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

c = venn2_circles(subsets=(1, 1, 1), color="#000000", alpha=1, linestyle="-.", linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
plt.title ("¡Mi diagrama de Venn!");
plt.show()
```



Cambiamos el estilo de la línea de uno de los círculos y le agregamos etiquetas con **set_labels** :

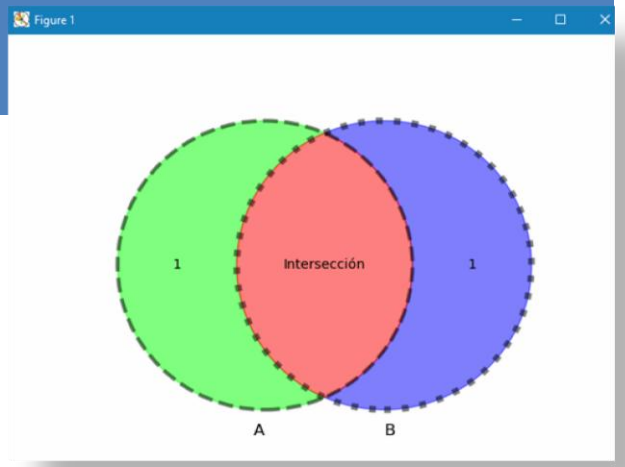
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

diagram = venn2((1, 1, 1), set_labels=('Grupo A', 'Grupo B'), set_colors=("#FF5733", "#68FF33"), alpha=1)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00“)
diagram.get_patch_by_id("11").set_color("#FF0000“)

c = venn2_circles(subsets=(1, 1, 1), color="#000000", alpha=0.5, linestyle="-.", linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
c[1].set_ls('dotted')
plt.title ("¡Mi diagrama de Venn!");
plt.show()
```



Ahora bien, lo que nos interesa es poder representar los valores. Estos pueden surgir de listas, tuplas, conjuntos , etc y como resultado de operaciones entre conjuntos. Ejemplo:

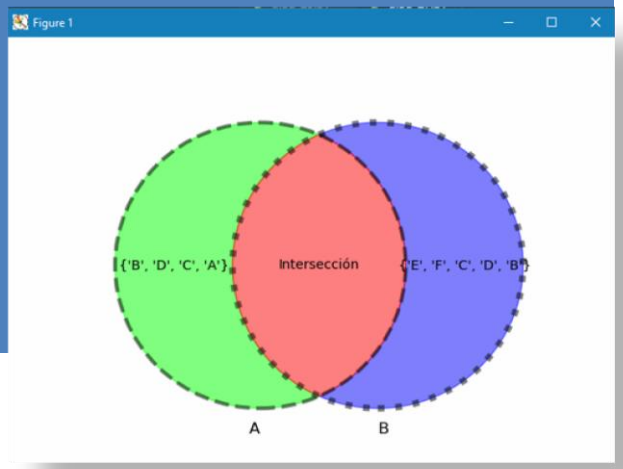
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

set1 = set(['A','B','C','D'])
set2 = set(['B','C','D','E','F'])

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=0.5)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

c = venn2_circles(subsets=(1, 1, 1), color="#000000", alpha=0.5, linestyle="dashed", linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
c[1].set_ls('dotted')
plt.show()
```

NO se visualizan cambios….



Para visualizar los valores vamos a pasarle como parámetros los sets a la propiedad **set_text()** de **get_label_by_id:**

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

set1 = set(['A','B','C','D'])
set2 = set(['B','C','D','E','F'])

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=0.5)
diagram.get_label_by_id("11").set_text("Intersección")
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

diagram.get_label_by_id('10').set_text(set1)
diagram.get_label_by_id('01').set_text(set2)

c = venn2_circles(subsets=(1, 1, 1), color="#000000",
                  alpha=0.5, linestyle="dashed",
                  linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
c[1].set_ls('dotted')
plt.show()
```



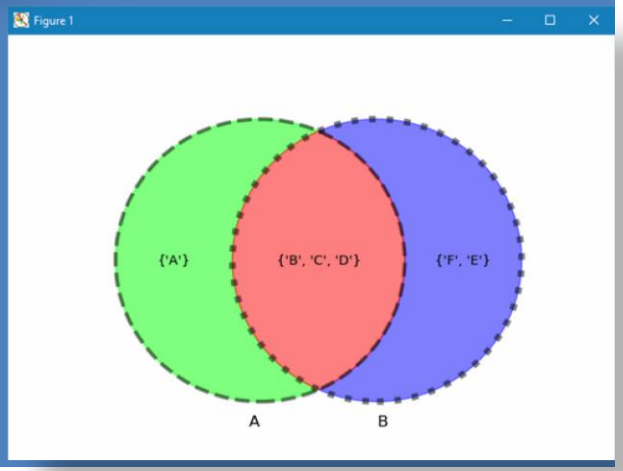
Este diagrama NO es lo que queríamos….

Vamos a eliminar la palabra “Intersección ” y a realizar algunas operaciones para mejorar nuestro gráfico:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

set1 = set(['A','B','C','D'])
set2 = set(['B','C','D','E','F'])
inter = set(set1.intersection(set2))
A = set1 - inter
B = set2 - inter

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=0.5)
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")

diagram.get_label_by_id('10').set_text(A)
diagram.get_label_by_id('01').set_text(B)
diagram.get_label_by_id('11').set_text(inter)

c = venn2_circles(subsets=(1, 1, 1), color="#000000",
          alpha=0.5, linestyle="dashed",
          linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
c[1].set_ls('dotted')
plt.show()
```

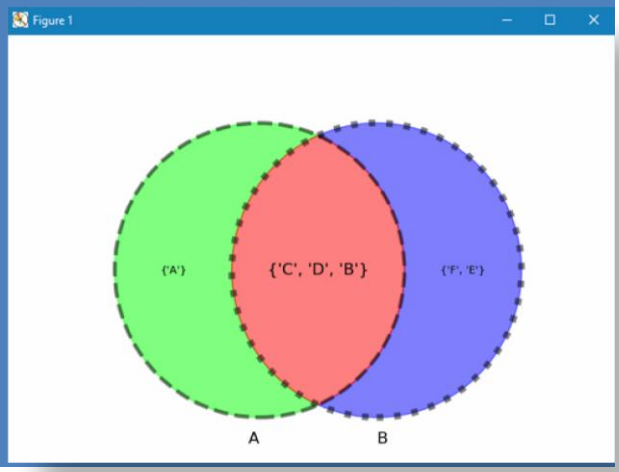
Realizamos operaciones entre conjuntos ….

Pasamos los resultados como parámetros



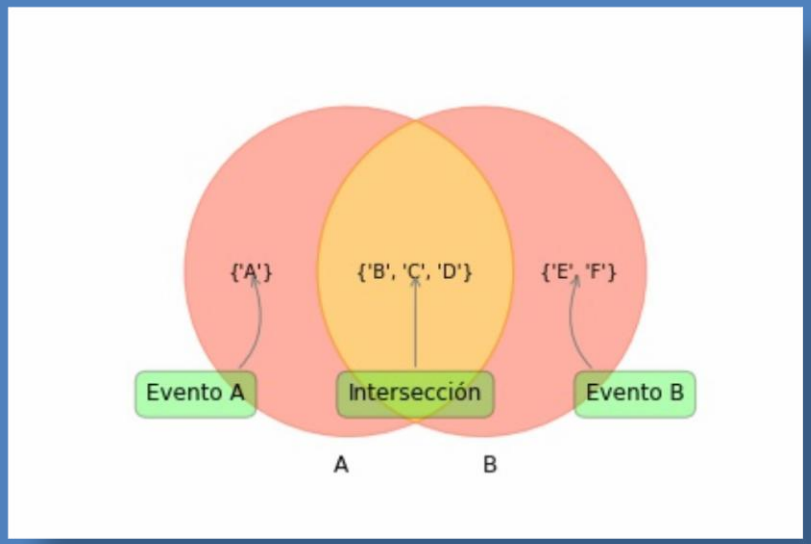
Destacamos la intersección cambiándole el tamaño de fuente con **set_fontsize**;

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn2, venn2_circles

set1 = set(['A','B','C','D'])
set2 = set(['B','C','D','E','F'])
inter = set(set1.intersection(set2))
A = set1 - inter
B = set2 - inter

diagram = venn2((1, 1, 1),set_colors=("#FF5733", "#68FF33"), alpha=0.5)
diagram.get_label_by_id("11").set_color("#000000")
diagram.get_patch_by_id("01").set_color("#0000FF")
diagram.get_patch_by_id("10").set_color("#00FF00")
diagram.get_patch_by_id("11").set_color("#FF0000")
diagram.get_label_by_id('10').set_text(A)
diagram.get_label_by_id('10').set_fontsize(8)
diagram.get_label_by_id('01').set_text(B)
diagram.get_label_by_id('01').set_fontsize(8)
diagram.get_label_by_id('11').set_text(inter)
diagram.get_label_by_id('11').set_fontsize(12)

c = venn2_circles(subsets=(1, 1, 1),
  color="#000000",
  alpha=0.5, linestyle="dashed", linewidth=3)
c[0].set_lw(3.0)
c[1].set_lw(5.0)
c[1].set_ls('dotted')
plt.show()
```



Podemos utillizar la función **annotate()** de matplotlib para agregar anotaciones:

```
from matplotlib_venn import *
from matplotlib import pyplot as plt

set1=set(['A','B','C','D'])
set2=set(['B','C','D','E','F'])
inter=set(set1.intersection(set2))
A=set1-inter
B=set2-inter

v = venn2(subsets = {'10': 1, '01': 1, '11': 1}, set_labels = ('A', 'B'))
v.get_patch_by_id('10').set_alpha(0.5)
v.get_patch_by_id('10').set_color('tomato')
v.get_patch_by_id('01').set_alpha(0.5)
v.get_patch_by_id('01').set_color('tomato')
v.get_patch_by_id('11').set_alpha(0.5)
v.get_patch_by_id('11').set_color('orange')
v.get_label_by_id('10').set_text(A)
v.get_label_by_id('01').set_text(B)
v.get_label_by_id('11').set_text(inter)
```

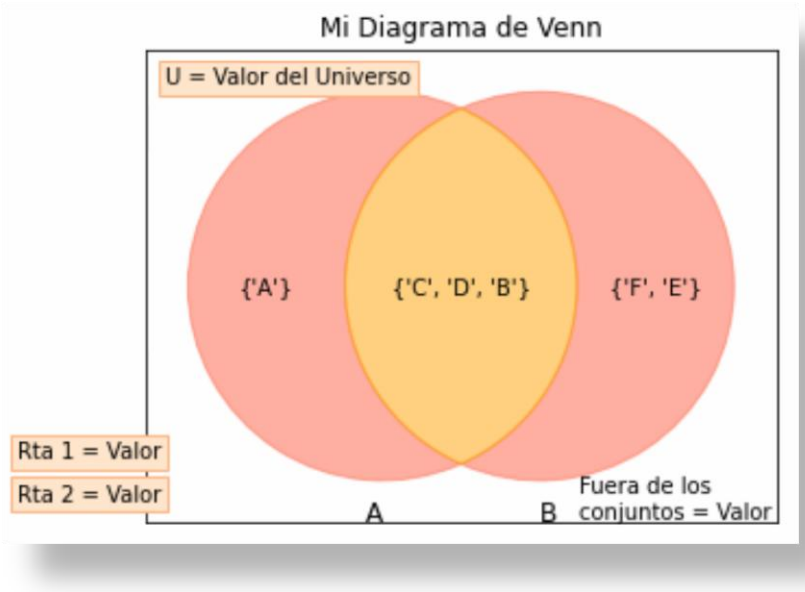



```
plt.annotate('Evento A', xy = v.get_label_by_id('10').get_position(), xytext = (-30,-70), size = 'large',
        ha = 'center', textcoords = 'offset points', bbox = dict(boxstyle = 'round, pad = 0.5', fc = 'lime', alpha = 0.3),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3, rad = 0.5', color = 'gray'))

plt.annotate('Evento B', xy = v.get_label_by_id('01').get_position(), xytext = (30,-70), size = 'large',
        ha = 'center', textcoords = 'offset points', bbox = dict(boxstyle = 'round, pad = 0.5', fc = 'lime', alpha = 0.3),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3, rad = -0.5',color = 'gray'))

plt.annotate('Intersección', xy = v.get_label_by_id('11').get_position(), xytext = (0,-70), size = 'large',
        ha = 'center', textcoords = 'offset points', bbox = dict(boxstyle = 'round, pad = 0.5', fc = 'lime', alpha = 0.3),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad = 0',color = 'gray'))

plt.show()
```

También podemos utillizar la función **plt.text()** para agregar anotaciones y **plt.axis()** para formar el recuadro :

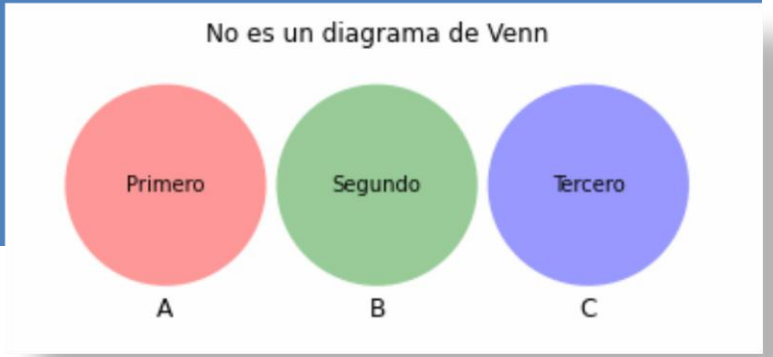
```
from matplotlib_venn import *
from matplotlib import pyplot as plt

set1=set(['A','B','C','D'])
set2=set(['B','C','D','E','F'])
inter=set(set1.intersection(set2))
A=set1-inter
B=set2-inter

v = venn2(subsets = {'10': 1, '01': 1, '11': 1}, set_labels = ('A', 'B'))
v.get_patch_by_id('10').set_alpha(0.5)
v.get_patch_by_id('10').set_color('tomato')
v.get_patch_by_id('01').set_alpha(0.5)
v.get_patch_by_id('01').set_color('tomato')
v.get_patch_by_id('11').set_alpha(0.5)
v.get_patch_by_id('11').set_color('orange')
v.get_label_by_id('10').set_text(A)
v.get_label_by_id('01').set_text(B)
v.get_label_by_id('11').set_text(inter)

plt.text(-1.05, -0.42,
      s="Rta 1 = " + str('Valor'), size=10,  ha="left",    va="bottom",
      bbox=dict(boxstyle="square",  ec=(1.0, 0.7, 0.5),
            fc=(1.0, 0.9, 0.8),))

plt.text(-1.05, -0.52,
      s="Rta 2 = " + str('Valor'),  size=10,  ha="left",    va="bottom",
      bbox=dict(boxstyle="square",   ec=(1.0, 0.7, 0.5),
            fc=(1.0, 0.9, 0.8),))
```

```
plt.text(-0.70, 0.52,
      s="U = " + str('Valor del Universo'),
      size=10,
      ha="left",
      va="top
      bbox=dict(boxstyle="square",  # tipo de cuadro
            ec=(1.0, 0.7, 0.5),
            fc=(1.0, 0.9, 0.8),))

# Valor del resto de los deportistas
plt.text(0.28, -0.55,
      s="Fuera de los\nconjuntos = " + str('Valor'),
      size=10)

plt.axis('on')
plt.title("Mi Diagrama de Venn")
plt.show()
```



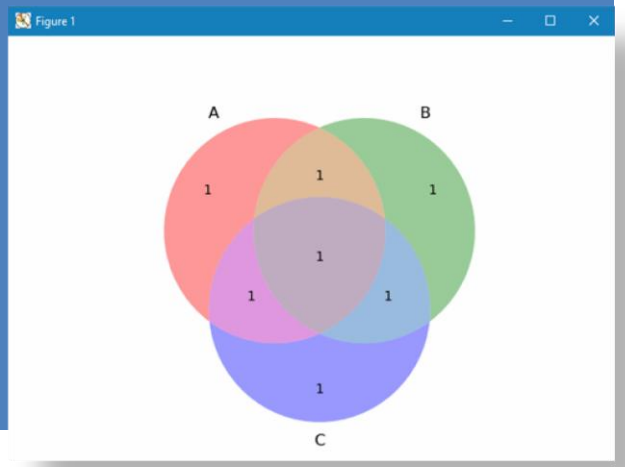
… Y si necesitamos representar más de 2 conjuntos?... Tenemos **venn3**

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3

v = venn3(subsets=(1,1,0,1,0,0,0))
v.get_label_by_id('100').set_text('Primero')
v.get_label_by_id('010').set_text('Segundo')
v.get_label_by_id('001').set_text('Tercero')
plt.title("No es un diagrama de Venn")
plt.show()
```



La función venn3() es esencialmente similar.
El **primer argumento será ahora una tupla de 7** elementos que se corresponden con los siete subconjuntos en el siguiente orden:
Abc, aBc, ABc, abC, AbC, aBC, ABC.

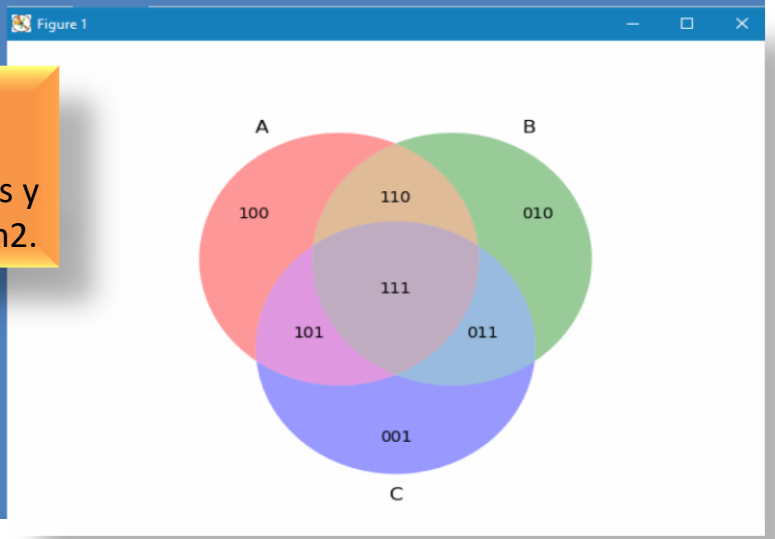
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3

diagram = venn3((1, 1, 1, 1, 1, 1, 1))
plt.show()
```



Para identificar a cada uno de los subconjuntos se emplean ahora tres dígitos según el orden A, B, C. La imagen que vamos a generar a continuación muestra cada uno de ellos con su respectivo identificador. Podemos establecer las etiquetas utilizando un ciclo for:

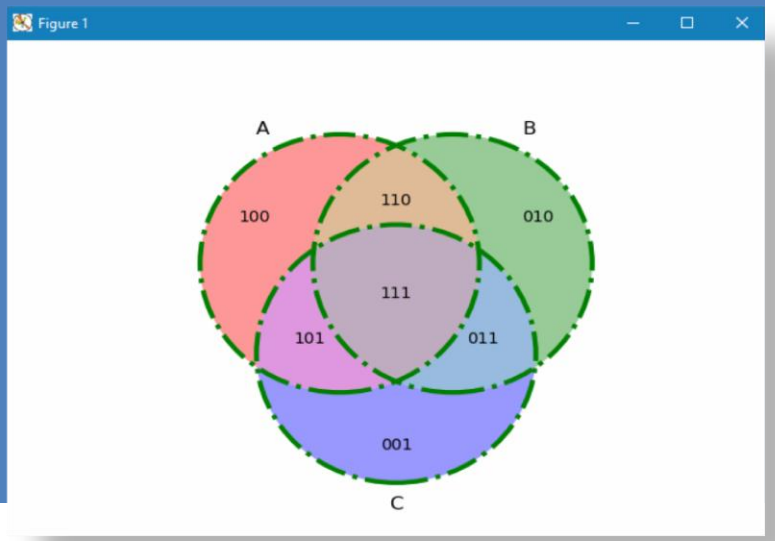
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3

diagram = venn3((1, 1, 1, 1, 1, 1, 1))

for subset in ("111", "110", "101", "100",
                                    "011", "010", "001"):
   diagram.get_label_by_id(subset).set_text(subset)
plt.show()
```

Para venn3 también aplican los atributos alpha, set_labels y set_colors, como en venn2.



Demarcamos los círculos como hicimos con venn2:

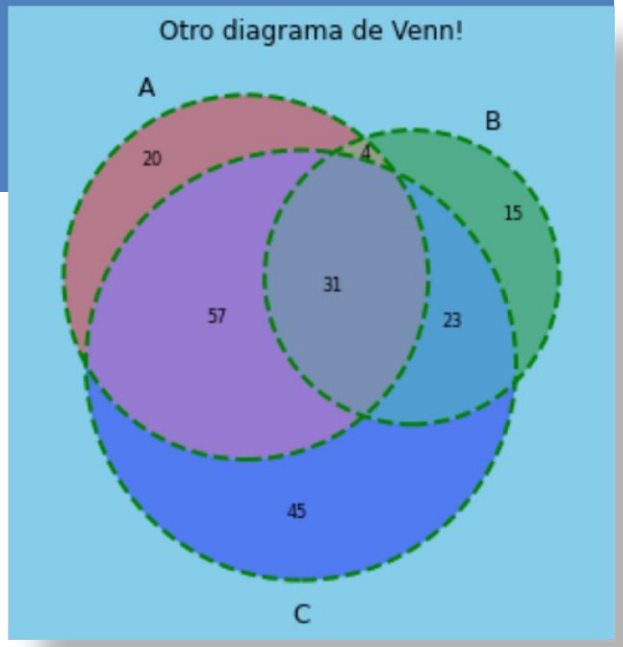
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3, venn3_circles

diagram = venn3((1, 1, 1, 1, 1, 1, 1))
for subset in ("111", "110", "101", "100", "011", "010", "001"):
   diagram.get_label_by_id(subset).set_text(subset)

venn3_circles(subsets=(1, 1, 1, 1, 1, 1, 1),
      color="#008000", alpha=1, linestyle="-.",
      linewidth=3)
plt.show()
```



Ahora distribuyo los valores, defino el tamaño de la ventana y color de fondo del gráfico, también le agrego un título:

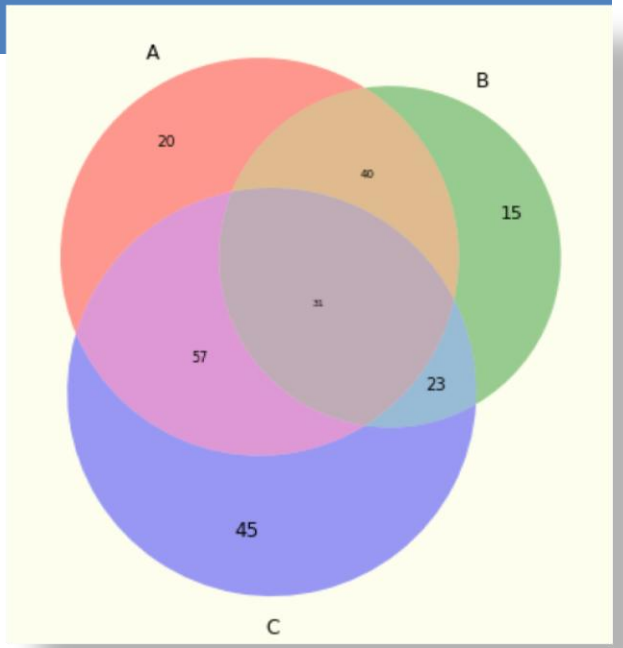
```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3, venn3_circles

plt.figure(figsize=(7,5), facecolor='skyblue')

diagram = venn3(subsets=(20, 15, 4, 45, 57, 23, 31), set_labels = ('A', 'B', 'C'))

for subset in ("111", "110", "101", "100", "011", "010", "001"):
    diagram.get_label_by_id(subset).set_fontsize(8)

venn3_circles(subsets=(20, 15, 4, 45, 57, 23, 31),
                  color="#008000", alpha=1,
                  linestyle='dashed',linewidth=2)
plt.title("Otro diagrama de Venn!")
plt.savefig("Mi_diagrama.jpg")
plt.savefig("Mi_diagrama.pdf")
plt.show()
```

Nota: para este ejemplo se usaron distintos valores en subsets para la mejor comprensión del diagrama.
Agregando la instrucción:
plt.savefig("<nombre>.<extensión>"), se puede generar un archivo pdf, png, o jpg de la imagen.



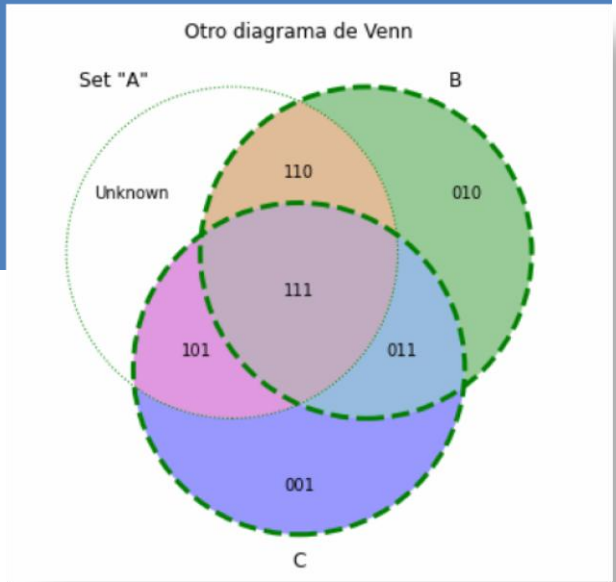
Podemos manejar el tamaño de la fuente de la siguiente manera:

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn3, venn3_circles

plt.figure(figsize=(7,5), facecolor='ivory')
diagram = venn3(subsets=(20, 15, 40, 45, 57, 23, 31), set_labels = ('A', 'B', 'C'))
cont=6
for subset in ("111", "110", "101", "100", "011", "010", "001"):
    diagram.get_label_by_id(subset).set_fontsize(cont)
    cont = cont+1

plt.show()
```



Si necesitamos representar un área desconocida, podemos hacerlo de la siguiente manera:

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3, venn3_circles

plt.figure(figsize=(4,4))
diagram = venn3(subsets=(1, 1, 1, 1, 1, 1, 1), set_labels = ('A', 'B', 'C'))

for subset in ("111", "110", "101", "100", "011", "010", "001"):
    diagram.get_label_by_id(subset).set_text(subset)

diagram.get_patch_by_id('100').set_alpha(1.0)
diagram.get_patch_by_id('100').set_color('white')
diagram.get_label_by_id('100').set_text('Unknown')
diagram.get_label_by_id('A').set_text('Set "A"')
c = venn3_circles(subsets=(1, 1, 1, 1, 1, 1, 1),
    color="#008000", alpha=1,
    linestyle='dashed',linewidth=3)
c[0].set_lw(1.0)
c[0].set_ls('dotted')
plt.title("Otro diagrama de Venn")
plt.show()
```



https://matplotlib.org/

Customizing Matplotlib with style sheets and rcParams — Matplotlib 3.4.1 documentation

https://pypi.org/project/matplotlib-venn/

Specifying Colors — Matplotlib 3.4.3 documentation

Choosing Colormaps in Matplotlib — Matplotlib 3.4.3 documentation

FIN