15 - Pyplot

December 1, 2022

1 Matplotlib

En este tópico se trabajará con un módulo que se complementa con `Numpy`, utilizado para graficar vectores. Este módulo es distinto a los anteriormente vistos, puesto que es el único módulo con elementos gráficos con el que se trabajará a lo largo de este curso.

`Matplotlib` es un módulo desarrollado para realizar gráficos tanto en 2D como en 3D. Al igual que `Numpy`, es de código abierto, distribuido de manera gratuita y que no es parte de la biblioteca estándar que posee Python, vale decir, requiere realizar una instalación adicional.

Debido a su versatilidad, el módulo `Matpotlib` permite realizar gráficos de distintos tipos, entre los cuales se pueden nombrar los gráficos de línea, histogramas, de torta, de barra, entre otros.

1.1 Instalación

Como se mencionó previamente, para poder utilizar las funcionalidades que ofrece `matplotlib` es necesario instalarlo. Para ello, se sugiere revisar el apunte de `numpy`, en donde se explica de manera detallada cómo instalar distintos módulos en el sistema.

1.2 Característica Generales

Dentro del módulo `matplotlib` existe un submódulo llamado `pyplot`, el cual permite a `Python` realizar funciones similares a las utilizadas por herramientas de programación científica como lo son Matlab y Octave, entre otros.

En este curso, nos enfocaremos en el uso del submódulo `pyplot`, puesto que ofrece las herramientas necesarias para generar distintos tipos de gráficas.

Como se mencionó previamente, `matplotlib` es un `módulo` que requiere instalación para que esté disponible para su uso en Python.

Una vez instalado, para poder utilizarlo, es necesario realizar la importación de funciones (como se vio en clases anteriores).

Dicha importación se realizará de la siguiente manera:

[1]:
```
import matplotlib.pyplot as plt
# De esta manera el sub-módulo pyplot será llamado como plt
# Cada vez que se requiera llamar una función de dicho submódulo, se antepondrá␣
 ↪la palabra plt
# de la forma: plt.<función> a utilizar
```

1.3 Gráficas en Python

Como se mencionó previamente, el sub-módulo pyplot ofrece distintas posibilidades de gráficas:

1. Gráficos lineales
2. Histogramas
3. Gráficos de Torta
4. Gráficos de Barras
5. Gráficos de Multiseries
6. Múltiples Gráficos

1.3.1 1. Gráficos lineales

Para generar gráficas lineales se utilizará la función `plot()`. Dicha función requiere como parámetro una lista o arreglo. De esta forma, el parámetro ingresado será considerado como los valores del **eje** y del gráfico y de forma automática, Python asumirá que los valores del **eje** x comenzarán desde el valor 0 e irán aumentando de una unidad (en otras palabras, que el *índice* del elemento es su posición en x).

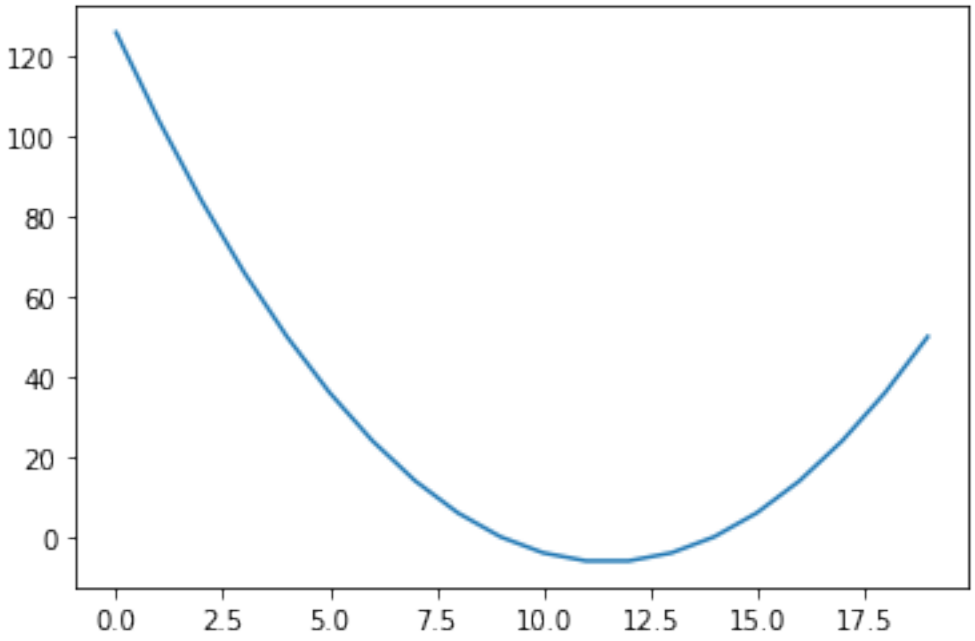
Para poder mostar el gráfico generado, el submódulo `pyplot` cuenta con una función denominada `show()`.

Por ejemplo, para graficar la curva

$$f(x) = x^2 - 3x - 4$$

podemos utilizar lo siguiente:

[8]:
```
# Ejemplo_1: Se graficará una curva utilizando la función plot() con solo 1␣
 ↪parámetro
# Importaciones
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL

# Procesamiento

# Se genera el vector que será utilizado como base para obtener el eje y
vector = np.arange(-10, 10)

# Se genera el eje y
funcion = vector**2 - vector*3 - 4

# Salida
# Como se mencionó en clases previas, un gráfico es un tipo de salida
# se genera el gráfico
plt.plot(funcion)

# Se solicita que se muestre por pantalla
plt.show()
```



Como se mencionó previamente, al utilizar un parámetro en la función `plot()`, Python asume que el eje x corresponde a su posición en el vector. Esto se ve reflejado en el **Ejemplo_1**.

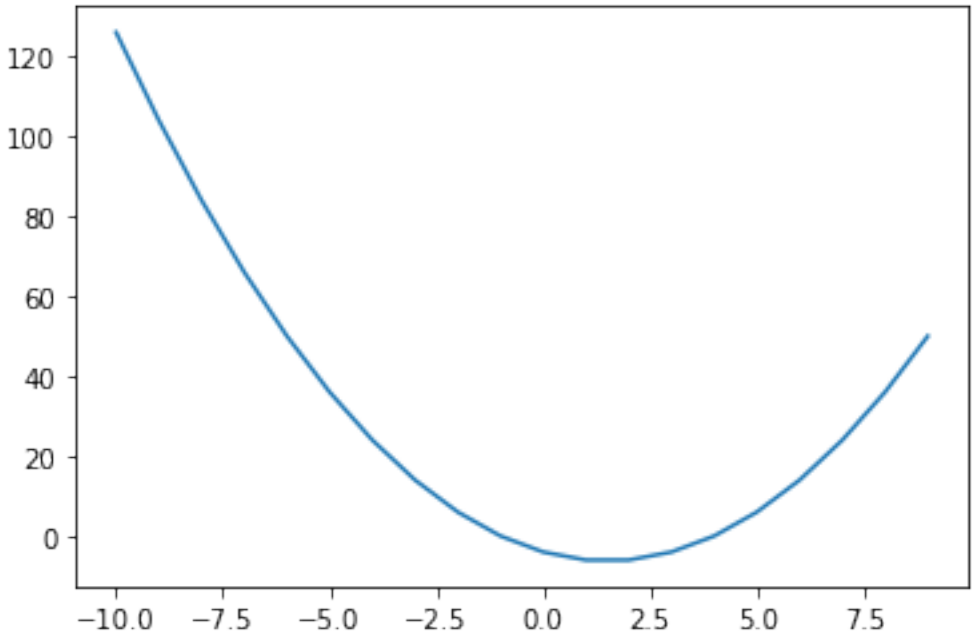
Ahora bien, la función `plot()` puede utilizar dos parámetros, los cuales son el eje x y el eje y, respectivamente. De esta manera, la función queda expresada de la siguiente forma:

```
plot(x, y)
```

Es muy importante que al momento de definir el eje x y el eje y de la curva, ambos tengan **la misma cantidad** de valores. Para ello, la forma más utilizada es obtener un valor de eje a partir del otro.

[10]:
```
# Ejemplo_2: Se graficará una curva utilizando la función plot() con 2 parámetros
# Importaciones
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
# Procesamiento
# Se genera el vector que será utilizado como base para obtener el eje y
vector_2 = np.arange(-10, 10)

# Se genera el eje y
funcion = vector_2**2 - vector_2*3 - 4

# Salida
# Se genera el gráfico
```

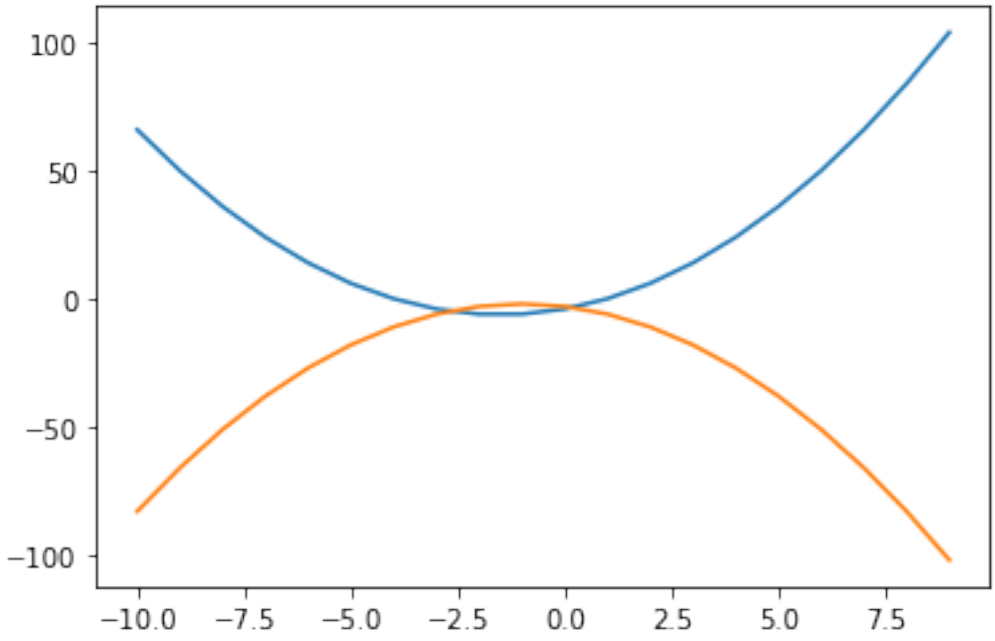
```
plt.plot(vector_2, funcion)

# Se solicita que se muestre por pantalla
plt.show()
```



El submódulo `pyplot` permite generar más de una curva de manera simultánea. Para ello, solo basta con generar la nueva curva antes de mostrar la gráfica (uso de la funcion `show()`):

[11]:
```
#Ejemplo_3: Se graficará 2 curvas utilizando la función plot() con 2 parámetros
# Importaciones
import numpy as np
import matplotlib.pyplot as plt

#BLOQUE PRINCIPAL
#Procesamiento
#Se genera el vector que será utilizado como base para obtener el eje y
vector_3 = np.arange(-10, 10)

#Se genera el eje y
funcion_3_A = vector_3**2 + vector_3*3 - 4
funcion_3_B = -vector_3**2 - 2*vector_3 - 3

#Salida
#se generan los gráficos
plt.plot(vector_3, funcion_3_A)
plt.plot(vector_3, funcion_3_B)
```

```
#Se solicita que se muestre por pantalla
plt.show()
```



Al observar los ejemplos entregados, se puede apreciar que las curvas están formadas por líneas contínua. Además, si se analiza el ejemplo_3, se ve claramente que al graficar 2 curvas, Python las interpreta o diferencia con colores distintos por defecto.

Como la idea es poder personalizar el trabajo realizado, el submódulo `pyplot` ofrece la opción de dar un **formato específico** a cada curva.

Personalizar Curva Cuando se requiere trabajar con 2 o más curvas en simultáneo es importante poder diferenciarlas una de otras, ya sea por su color o tipo de línea que une los puntos.

Es por ello que `pyplot` cuenta con la función `setp()`, la cual permite especificar los siguientes parámetros: * `marker`: define como se verán los puntos. * `linestyle`: define como se verá la línea que une los puntos * `color`: define el color de la línea y los puntos generados. * `linewidth`: define el grosor de la línea (número flotante que representa el grosor en puntos).

Para utilizar la función `setp()` en cada una de las curvas requeridas, es necesario asignarlas a una varible al momento de crearlas.

Ejemplo:

```
grafico_1 = plt.plot(vectorX_1, vector_Y_1)
grafico_2 = plt.plot(vectorX_2, vector_Y_2)
```

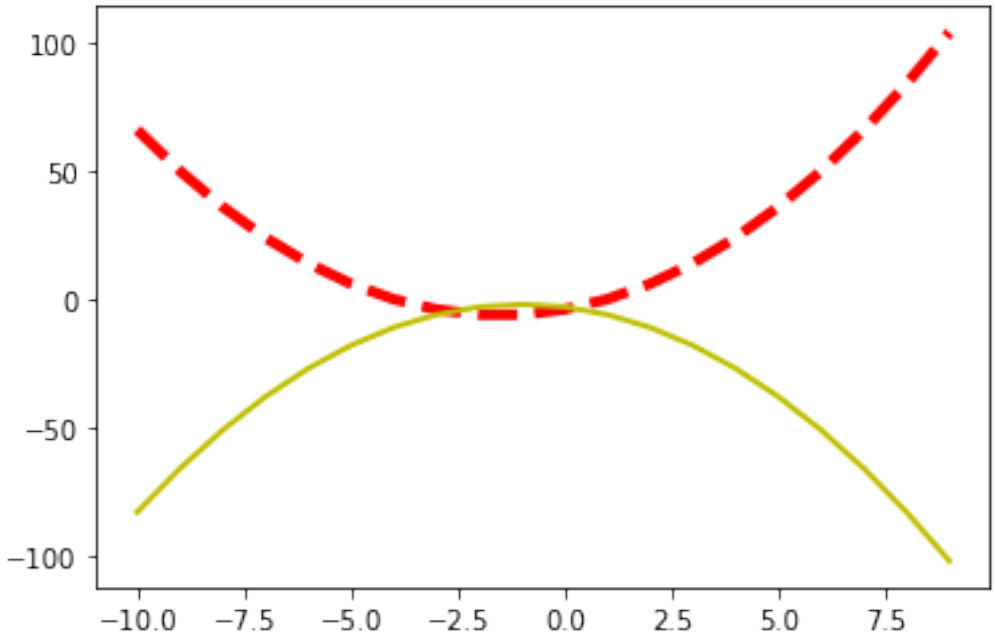
Una vez que se tengan claras las características que se quieran dar a las curvas, se puede llamar a la función de cualquiera de las siguientes dos maneras:

```
plt.setp(<nombre del gráfico>, "marker", "*", "linestyle", "--", "color", r", "linewidth", 2)
plt.setp(<nombre del gráfico>, marker="*", linestyle="--", color="r", linewidth=2)
```

En el caso anterior, se está indicando que, para un determinado gráfico, se utilizarán estrellas para marcar la ubicación de cada punto (*), con estilo de línea segmentada ("--") de color rojo ("r") y el grosos de línea 2.0 puntos.

Utilicemos la función `setp()` en el ejemplo anterior para personalizarlo:

[15]:
```
# Ejemplo_4: Se graficará 2 curvas utilizando la función plot() con 2 parámetros
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL

# Procesamiento
# Se genera el vector que será utilizado como base para obtener el eje y
vector_3 = np.arange(-10, 10)

# Se genera el eje y
funcion_3_A = vector_3**2 + vector_3*3 - 4
funcion_3_B = -vector_3**2 - 2*vector_3 - 3

# Salida
# Se generan los gráficos
grafico1 = plt.plot(vector_3, funcion_3_A)
grafico2 = plt.plot(vector_3, funcion_3_B)

# Se dan las características a cada gráfico, usando ambos métodos (sin␣
 ↪mezclarlos)
# Línea segmentada, color rojo ([r]ed), ancho 4 puntos
plt.setp(grafico1, "linestyle", "--", "color", "r", "linewidth", 4)
# Línea continua, color amarillo ([y]ellow), ancho 2 puntos
plt.setp(grafico2, linestyle="-", color="y", linewidth=2)

# Se solicita que se muestre por pantalla
plt.show()
```



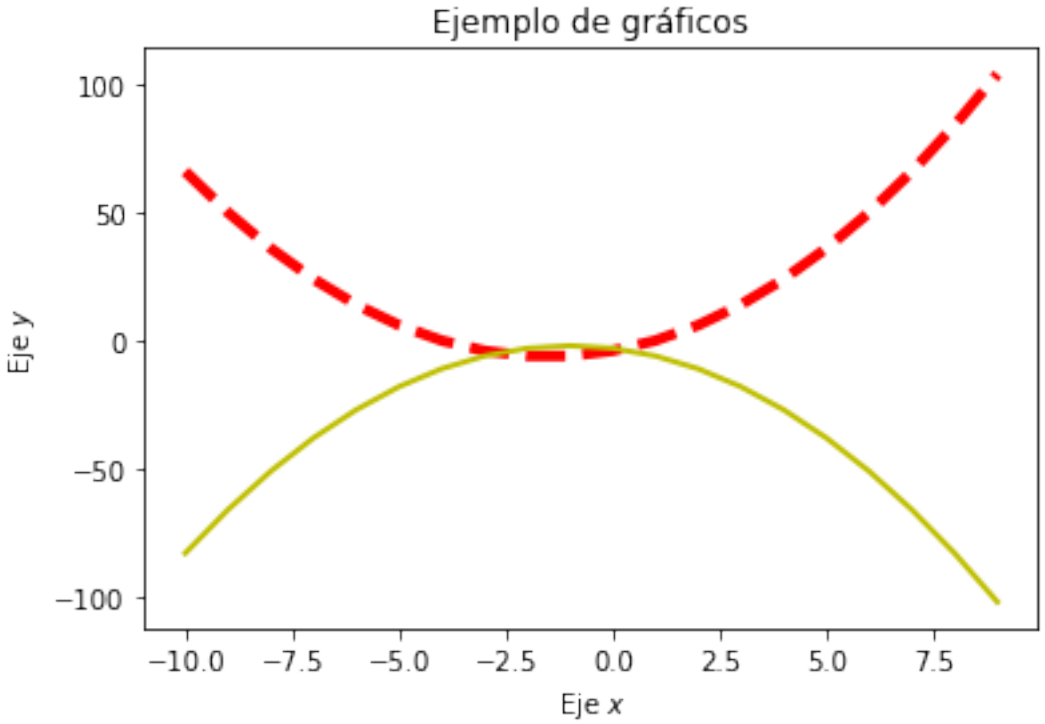
Además de las características que se le pueden agregar a las curvas, también existe la posibilidad de agregar detalles al trabajo realizado.

Para ello, se puede personalizar indicando el **título de la gráfica**, qué es el **eje** x y a qué corresponde el **eje** y. * Para agregar el título se utilizará la siguiente línea de código `plt.title(<título del gráfico>)`. * Para agregar el nombre del eje X se utilizará la siguiente línea de código `plt.xlabel(<nombre del eje x>)`. * Para agregar el nombre del eje Y se utilizará la siguiente línea de código `plt.ylabel(<nombre del eje y>)`.

Utilicemos los métodos vistos recientemente para personalizar aún más el gráfico:

[17]:
```
#Ejemplo_5: Se graficará 2 curvas utilizando la función plot() con 2 parámetros
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL

# Procesamiento
# Se genera el vector que será utilizado como base para obtener el eje y
vector_3 = np.arange(-10, 10)

# Se genera el eje y
funcion_3_A = vector_3**2 + vector_3*3 - 4
funcion_3_B = -vector_3**2 - 2*vector_3 - 3

# Salida
# Se generan los gráficos
```

```
grafico1 = plt.plot(vector_3, funcion_3_A)
grafico2 = plt.plot(vector_3, funcion_3_B)

# Se dan las características a cada gráfico, usando ambos métodos (sin␣
 ↪mezclarlos)
# Línea segmentada, color rojo ([r]ed), ancho 4 puntos
plt.setp(grafico1, "linestyle", "--", "color", "r", "linewidth", 4)
# Línea continua, color amarillo ([y]ellow), ancho 2 puntos
plt.setp(grafico2, linestyle="-", color="y", linewidth=2)

#Se dan las características del gráfico
plt.title("Ejemplo de gráficos")
plt.xlabel("Eje $x$")
plt.ylabel("Eje $y$")

#Se solicita que se muestre por pantalla
plt.show()
```



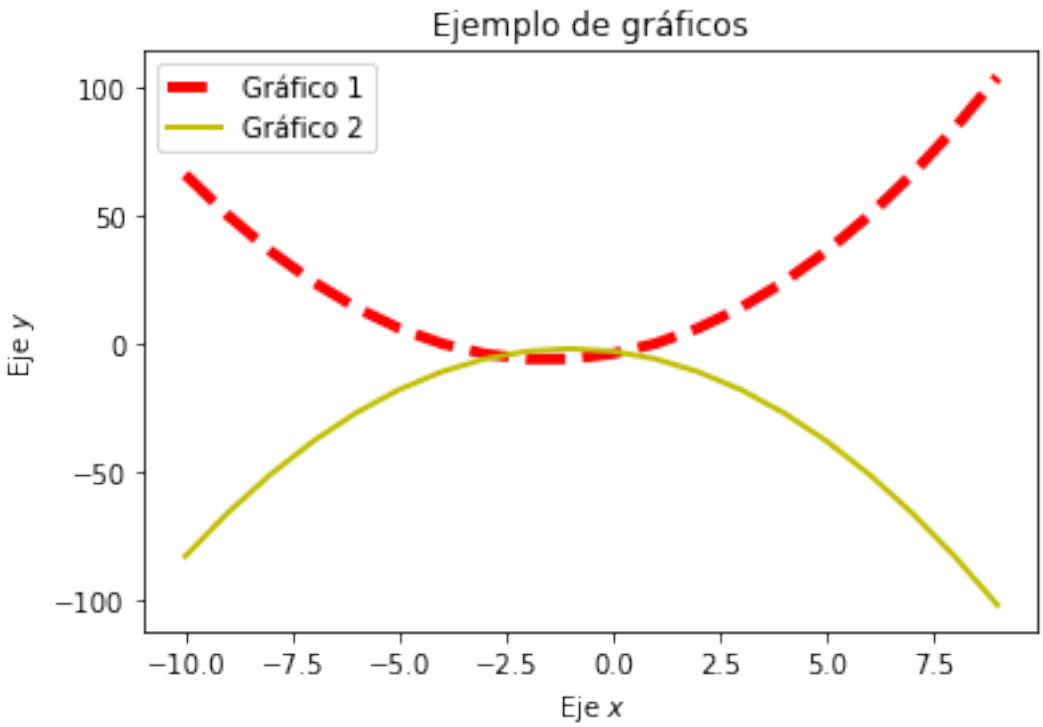
Ahora que se tiene un gráfico con muchos más detalles, se puede observar que existen 2 curvas pero no se puede distinguir a simple vista qué representa cada una.

Para solucionar la problemática anterior, se añadirá una leyanda, para indicar a qué función corresponde cada recta. Para poder agregar lo anterior, el submódulo `pyplot` posee el método `.legend()`, el cual permite mostrar el nombre asociado a cada recta.

Lo anterior solo es aplicable si se modifica un parte del código, o visto de otra forma, se agrega `label =<nombre de curva>`, ya sea dentro de la función `plot()` o dentro del método `.setp()`.

Utilicemos los métodos vistos recientemente para personalizar aún más el gráfico:

[19]:
```
#Ejemplo_6: Se graficará 2 curvas utilizando la función plot() con 2 parámetros
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL

# Procesamiento
# Se genera el vector que será utilizado como base para obtener el eje y
vector_3 = np.arange(-10, 10)

# Se genera el eje y
funcion_3_A = vector_3**2 + vector_3*3 - 4
funcion_3_B = -vector_3**2 - 2*vector_3 - 3

# Salida
# Se generan los gráficos
grafico1 = plt.plot(vector_3, funcion_3_A, label="Gráfico 1")
grafico2 = plt.plot(vector_3, funcion_3_B, label="Gráfico 2")

# Se dan las características a cada gráfico, usando ambos métodos (sin␣
 ↪mezclarlos)
# Línea segmentada, color rojo ([r]ed), ancho 4 puntos
plt.setp(grafico1, "linestyle", "--", "color", "r", "linewidth", 4)
# Línea continua, color amarillo ([y]ellow), ancho 2 puntos
plt.setp(grafico2, linestyle="-", color="y", linewidth=2)

#Se dan las características del gráfico
plt.title("Ejemplo de gráficos")
plt.xlabel("Eje $x$")
plt.ylabel("Eje $y$")

#Se da la instrucción para mostrar la leyenda
plt.legend()

#Se solicita que se muestre por pantalla
plt.show()
```



1.3.2 2. Histogramas

Los histogramas son un tipo de gráfico que represeta la distribución de un conjunto de datos, en forma de barras.

Su uso se basa en generar una visión general o panorámica de la distribución de la población o muestra respectiva.

Para realizar este tipo de gráfico, la biblioteca matplotlib ofrece el método `.hist()`.

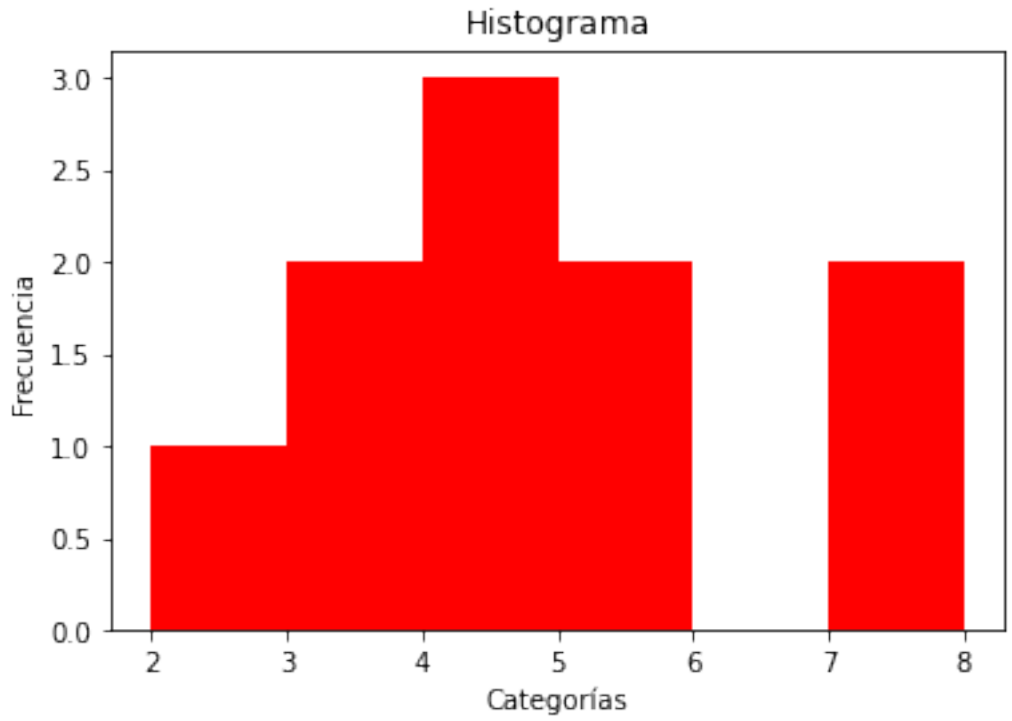
Para graficar histogramas, el método `.hist()` ofrece diversas características entre las cuales destacan: * Frecuencia o datos de un evento * Cantidad de eventos o categorías * Color de las barras

`plt.hist(<frecuencia>, bins=<cantidad categorías>, color=<color>`

A continuación se muestra la generación de un Histograma con un ejemplo concreto.

[21]:
```
# Ejemplo_7: Se graficará un Histograma
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los datos a graficar (pueden ser en formato lista como vector)
datos = [3, 4, 2, 3, 4, 5, 4, 7, 8, 5]
```

```
# SALIDA
# Se genera el Histograma indicando la generacióin de 6 categorías y color rojo
histograma = plt.hist(datos, bins=6, color = "r")

# Se dan las características del gráfico
plt.xlabel("Categorías")
plt.ylabel("Frecuencia")
plt.title("Histograma")

# Se solicita que se muestre por pantalla
plt.show()
```

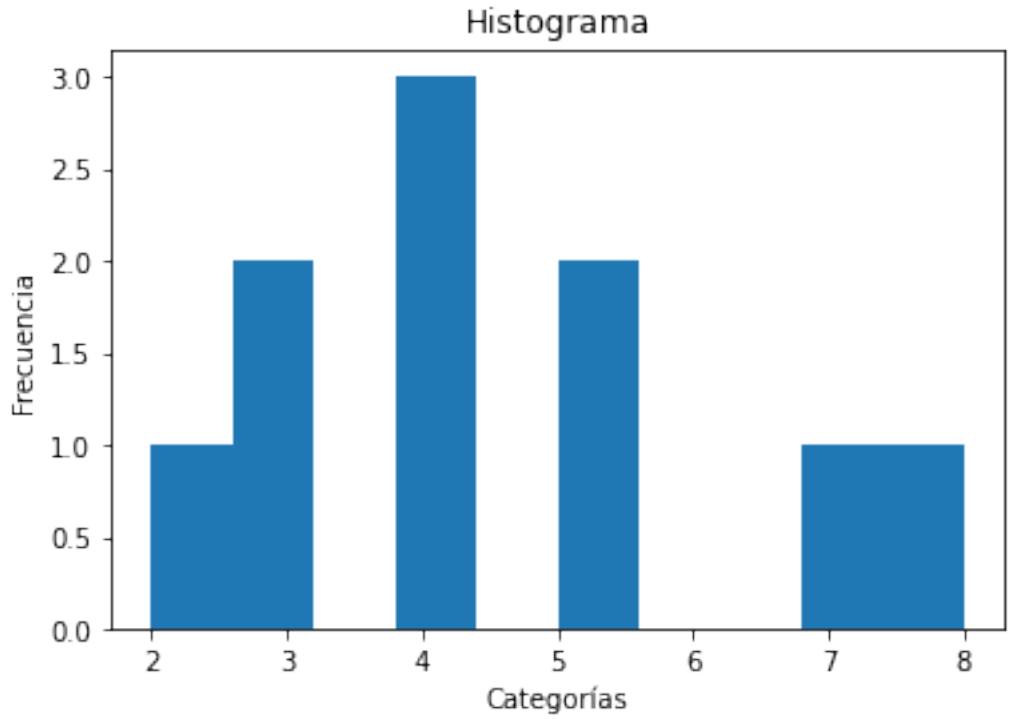


El **Ejemplo_7** muestra la generación de un Histograma con datos dados como lista. Adicionalmente, se indica la cantidad de categorías que se requiere y un colo en específico para las barras de la gráfica.

No obstante, para la generación de un Histograma solo se necesita, como mínimo, la frecuencia. Si no se especifican las categorías, estás por defecto serán 10 y el color será azul. Lo anterior se ve reflejado en el **Ejemplo_8**.

[22]:
```
# Ejemplo_8: Se graficará un Histograma
import numpy as np
import matplotlib.pyplot as plt
```

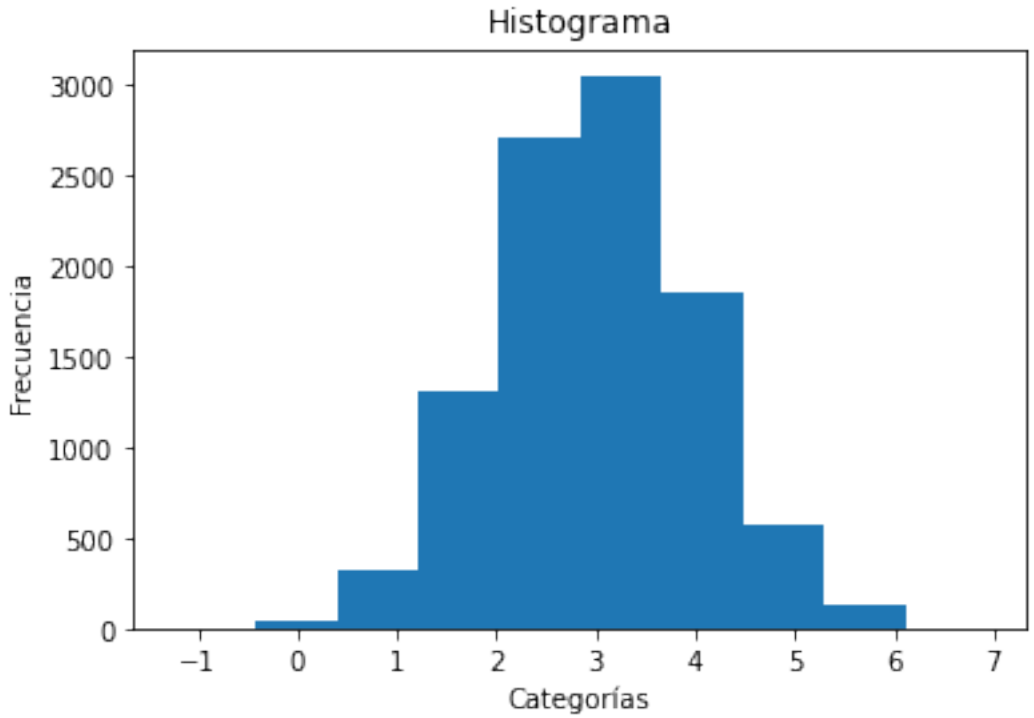
```
# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los datos a graficar (pueden ser en formato lista como vector)
datos = [3, 4, 2, 3, 4, 5, 4, 7, 8, 5]

# SALIDA
# Se genera el Histograma
histograma = plt.hist(datos)

# Se dan las características del gráfico
plt.xlabel("Categorías")
plt.ylabel("Frecuencia")
plt.title("Histograma")

# Se solicita que se muestre por pantalla
plt.show()
```



[23]:
```
# Un ejemplo usando un vector generado aleatoriamente
# Importaciones
import numpy as np
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
```

```
# Procesamiento
# Generación de valores aleatorios: una distribución normal con media 3 y␣
 ↪dispersión 1,
# con diez mil valores
datos = np.random.normal(3, 1, size=10000)

# SALIDA
# Se genera el Histograma
histograma = plt.hist(datos)

# Se dan las características del gráfico
plt.xlabel("Categorías")
plt.ylabel("Frecuencia")
plt.title("Histograma")

# Se solicita que se muestre por pantalla
# plt.show()
```

[23]: Text(0.5, 1.0, 'Histograma')



1.3.3 3. Gráficos de Torta

Un gráfico de torta, pastel o pie es un gráfico circular que está dividido en secciones, mostrando de manera pictórica la distribución de cada categoría según su composición.

Alguna de las características/ventajas que ofrece este tipo de gráfico son: * Por lo general, los datos son entregados en porcentaje * Se utiliza para comparar variables

Para realizar este tipo de gráfico, la biblioteca Matplotlib ofrece el método `.pie()`.

Para generar el gráfico de Torta, el método `.pie()` necesita, por lo menos contar con la siguiente información: * Datos numéricos para repartir o distribuir la torta * Nombre de las categorías

`plt.pie(<Datos numéricos>, labels=<lista de nombres de categorías>`

A continuación, se muestra la generación de un gráfico de Torta con un ejemplo concreto.

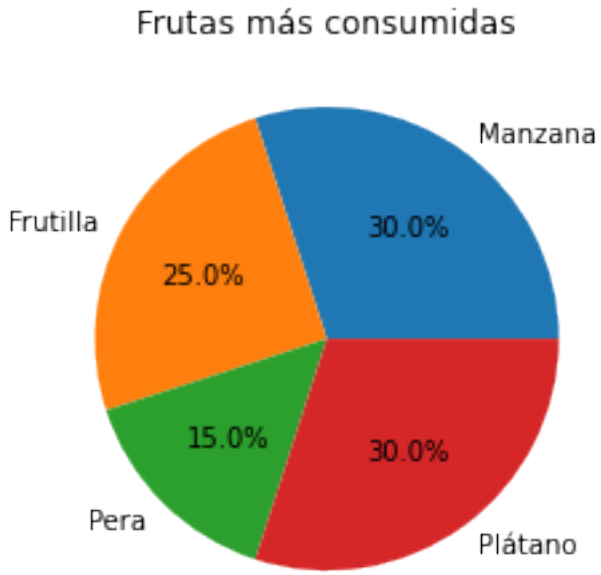
[24]:
```
# Ejemplo_9: Se graficará una torta con datos de frutas más consumidas
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las cantidades de personas que consumen un tipo determinado de␣
 ↪frutas
eleccion_frutas = [30, 25, 15, 30]

# Se definen las furtas escogidas (misma cantidad de elementos en las etiquetas
# que en los datos)
frutas = ["Manzana", "Frutilla", "Pera", "Plátano"]

# SALIDA
# Se genera el gráfico de Torta
grafico_torta = plt.pie(eleccion_frutas, labels=frutas)

# Se da nombre a la gráfica
plt.title("Frutas más consumidas")

# Se solicita que se muestre por pantalla
plt.show()
```

El **Ejemplo_9** muestra la distribución de frutas consumidas por determinadas personas. No obstante, no aparecen la cantidad numérica de cada elección.

El **Ejemplo_10** ilustra la situación planteada, en donde se utilizará el argumento `autopct="%0.1f%%"` que indica que muestre los datos, en formato procentaje, con 1 decimal.

[25]:
```
# Ejemplo_10: Se graficará una torta con datos de frutas más consumidas
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las cantidades de personas que consumen un tipo determinado de␣
 ↪frutas
eleccion_frutas = [30, 25, 15, 30]

# Se definen las furtas escogidas (misma cantidad de elementos en las etiquetas
# que en los datos)
frutas = ["Manzana", "Frutilla", "Pera", "Plátano"]

# SALIDA
# Se genera el gráfico de Torta
grafico_torta = plt.pie(eleccion_frutas, labels=frutas, autopct="%0.1f%%")

# Se da nombre a la gráfica
plt.title("Frutas más consumidas")
```

```
# Se solicita que se muestre por pantalla
plt.show()
```

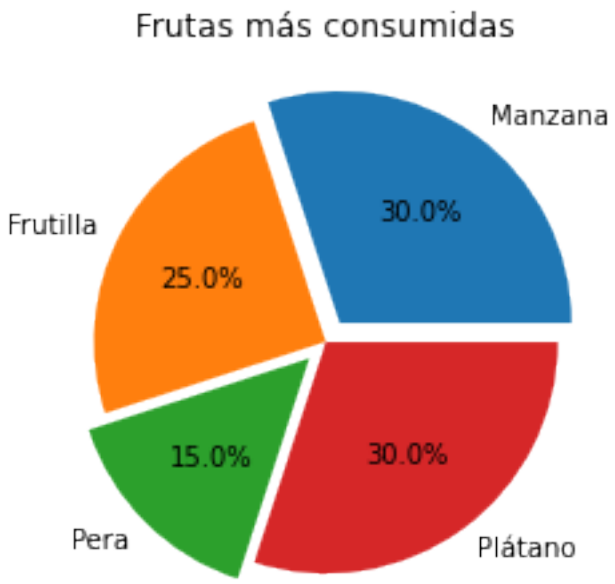


Si se desea agregar una separación entre alguna o más las partes de la gráfica, se utiliza ela rgumento `explode`. Este corresponde a una lista de tantos valores como datos a graficar que indican qué proporción del radio de la torta se separará cada valor del centro de esta. Por ejemplo, si tenemos que el parámetro `explode` será `[0.1, 0, 0.1, 0]`, estamos indicando que el primer y tercer dato estarán separados del centro, a una distancia de 0.1 veces el radio de la torta.

El **Ejemplo_11** ilustrará la gráfica con separación de algunas partes:

[26]:
```
#Ejemplo_11: Se graficará una torta con datos de frutas más consumidas
import matplotlib.pyplot as plt

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las cantidades de personas que consumen un tipo determinado de␣
 ↪frutas
eleccion_frutas = [30, 25, 15, 30]

# Se definen las furtas escogidas (misma cantidad de elementos en las etiquetas
# que en los datos)
frutas = ["Manzana", "Frutilla", "Pera", "Plátano"]

# La separación
separacion = [0.1, 0, 0.1, 0]
```

```
# SALIDA
# Se genera el gráfico de Torta
grafico_torta = plt.pie(eleccion_frutas, labels=frutas, autopct="%0.1f%%",␣
 ↪explode=separacion)

# Se da nombre a la gráfica
plt.title("Frutas más consumidas")

# Se solicita que se muestre por pantalla
plt.show()
```



Como información general en este tipo de gráfico, los valores entregados no necesariamente deben sumar 100, pues el mismo gráfico se encarga de calcular las proporciones. Además, si se define las categorías y los valores de ésta, ambas variables deben tener la misma cantidad de elementos.

1.3.4 4. Gráficos de Barras

Es un tipo de gráfico que muestra la información con barras rectangulares cuya altura o largo es proporcional a los valores que representan. Es utilizado para realizar comparaciones en categorías discretas, donde uno de los ejes muestra las categorías y el otro, los valores de esta.

Sus dos tipos son de barras verticales y horizontales, donde el más utilizado es el primero.

4.1. Gráficos de Barras Verticales En este tipo de gráficas, se utiliza la siguiente distribución: * En el eje horizontal (eje de las abscisas), se colocan las variables o grupos de estudio. * En el

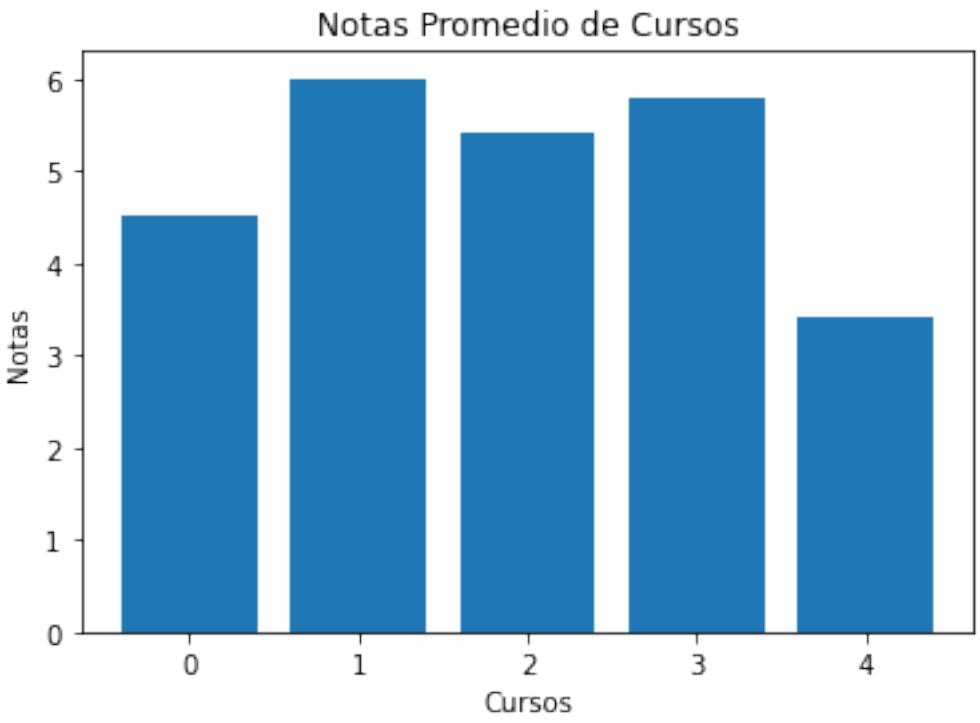
eje vertical (eje de las ordenadas) se distribuyen los valores que representan la frecuencia de las variables o grupos de estudio.

Cabe señalar que todas las barras tienen el mismo ancho.

Para realizar este tipo de gráfico en Python, la biblioteca Matplotlib ofrece el método `.bar()`.

Para generar el gráfico de barra vertical, el método `.bar()` necesita contar, por lo menos, con la siguiente información: * Posiciones (x): datos que representen las variables o grupos de estudio * Alturas (`height`): valores numéricos que representa la frecuencia o la cantidad a comparar

[28]:
```
# Ejemplo_12: Se graficarán barras veticales con datos de notas de un curso en una asignatura dada
# Importaciones
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las notas promedio de 5 cursos en una asignatura en particular (eje y)
notas = [4.5, 6.0, 5.4, 5.8, 3.4]

# Se definen las posiciones en el eje x
posiciones = np.arange(len(notas))

# SALIDA
# Se genera el gráfico
grafico_barra_vertical = plt.bar(posiciones, notas)

# Se da nombre a la gráfica
plt.title("Notas Promedio de Cursos")

# Se dan los nombres a los ejes
plt.xlabel("Cursos")
plt.ylabel("Notas")

# Se solicita que se muestre por pantalla
plt.show()
```

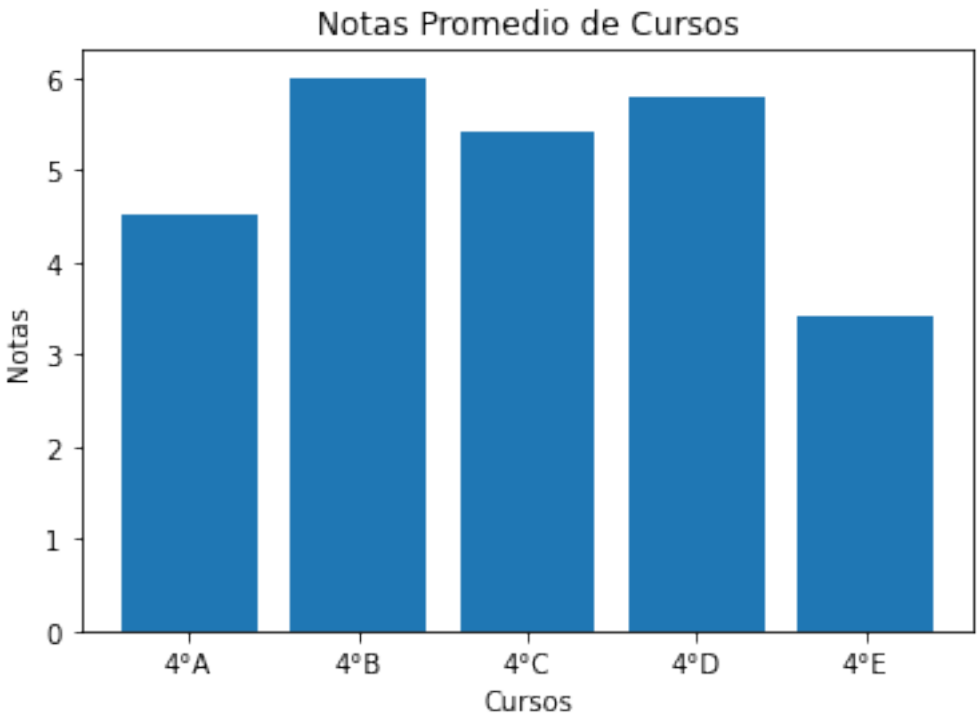


Se puede personalizar la gráfica del **Ejemplo_12** mediante la incorporación de etiquetas que especifiquen a qué corresponde cada valor del eje x. Para ello, se utilizará el método `.xticks()`. Los parámetros de este método son dónde va cada etiqueta (lista de valores del eje x) y cuáles son las etiquetas (lista de valores que mostrar, los cuales pueden ser números o strings).

Esto se muestra en el **Ejemplo_13**:

[30]:
```
# Ejemplo_12: Se graficarán barras veticales con datos de notas de un curso en una asignatura dada
# Importaciones
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las notas promedio de 5 cursos en una asignatura en particular (eje y)
notas = [4.5, 6.0, 5.4, 5.8, 3.4]

# Se definen las posiciones en el eje x
posiciones = np.arange(len(notas))

# Se definen los cursos (eje x)
cursos = ["4°A", "4°B", "4°C", "4°D", "4°E"]
```

```
# SALIDA
# Se genera el gráfico
grafico_barra_vertical = plt.bar(posiciones, notas)

# Se da nombre a la gráfica
plt.title("Notas Promedio de Cursos")

# Se dan los nombres a los ejes
plt.xlabel("Cursos")
plt.ylabel("Notas")

# Los cursos corresponden a etiquetas y señalamos en qué posición va cada una de␣
 ↪ellas
plt.xticks(posiciones, cursos)

# Se solicita que se muestre por pantalla
plt.show()
```



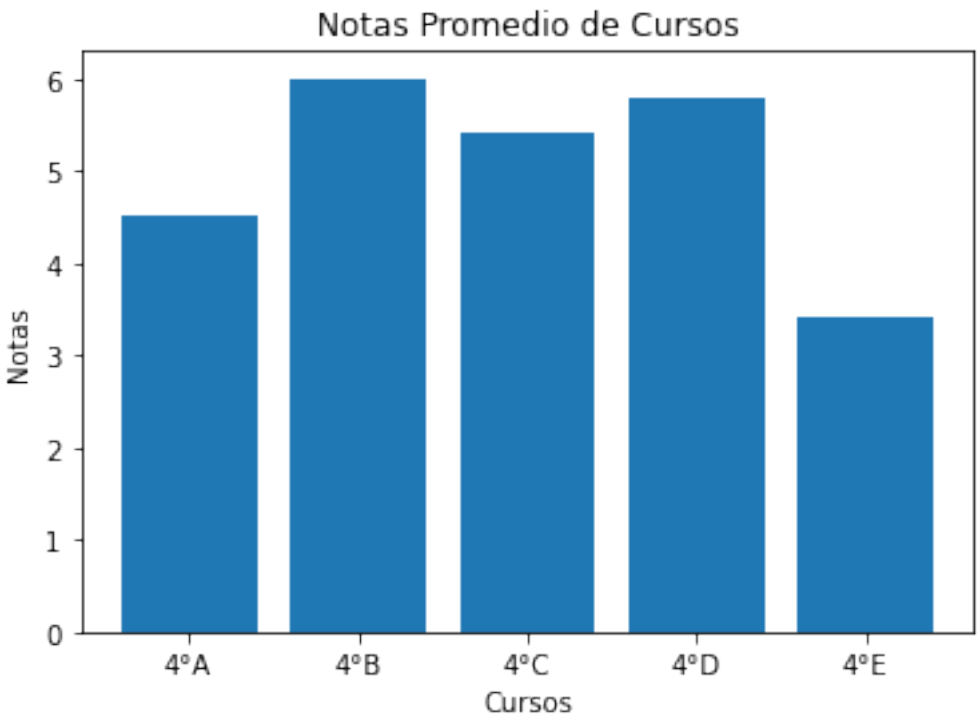
Nótese que la función `xticks` permite definir qué marcas se mostrarán en el eje en particular, por lo que sirven para cualquier gráfico con ejes (es decir, para el de torta, no, pero sí para los demás vistos acá).

Existe otra manera de poder colocar las etiquetas al exe x sin utilizar algún método en particular.

Para ello, se especificará de manera directa cuál es el eje x (nombre de los cursos), pero se debe tener cuidado que la cantidad de elementos del eje x debe ser la misma a la cantidad de elementos del eje y.

Esto se ejemplifica en el **Ejemplo_13**.

[32]:
```
# Ejemplo_13: Se graficarán barras veticales con datos de notas de un curso en una asignatura dada
# Importaciones
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las notas promedio de 5 cursos en una asignatura en particular (eje y)
notas = [4.5, 6.0, 5.4, 5.8, 3.4]

# Se definen las posiciones en el eje x
posiciones = np.arange(len(notas))

# SALIDA
# Se genera el gráfico
grafico_barra_vertical = plt.bar(cursos, notas)

# Se da nombre a la gráfica
plt.title("Notas Promedio de Cursos")

# Se dan los nombres a los ejes
plt.xlabel("Cursos")
plt.ylabel("Notas")

# Se solicita que se muestre por pantalla
plt.show()
```

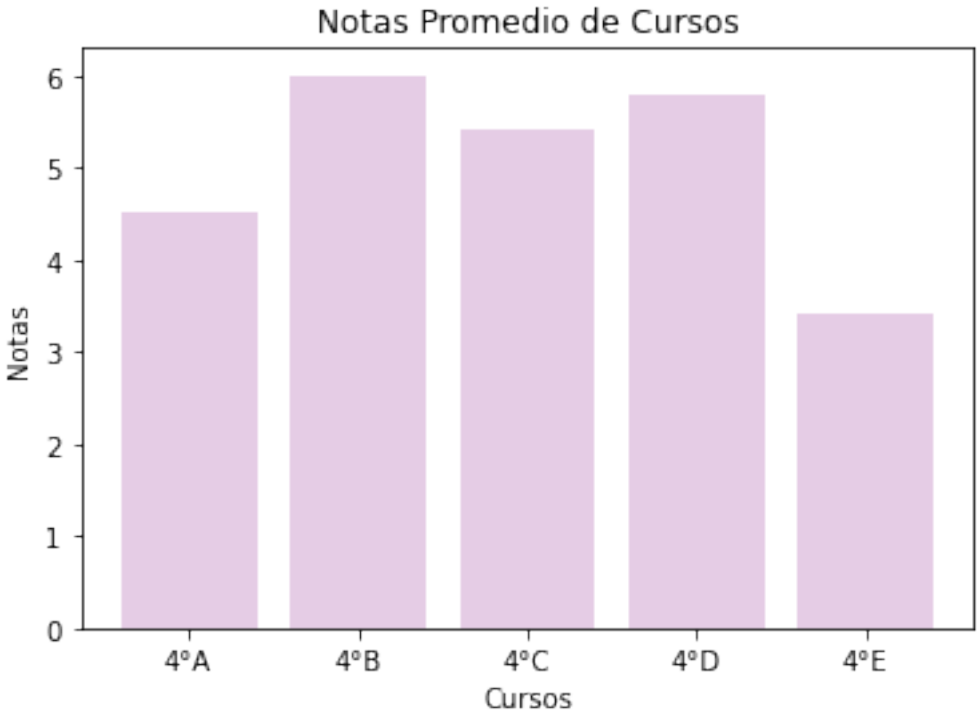


A este tipo de gráfico se le puede cambiar el color y suavizarlo según se requiera. Para ello, se utilizan las opciones `color` y `.alpha`, las cuales se adicionan dentro del método `.bar()`**.

El Ejemplo_14 ilustra lo anteriormente descrito:

[33]:
```
# Ejemplo_14: Se graficarán barras veticales con datos de notas de un curso en una asignatura dada
# Importaciones
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las notas promedio de 5 cursos en una asignatura en particular (eje y)
notas = [4.5, 6.0, 5.4, 5.8, 3.4]

# Se definen las posiciones en el eje x
posiciones = np.arange(len(notas))

# Se definen los cursos (eje x)
cursos = ["4°A", "4°B", "4°C", "4°D", "4°E"]

# SALIDA
```

```
# Se genera el gráfico
grafico_barra_vertical = plt.bar(posiciones, notas, color="purple", alpha=0.2)

# Se da nombre a la gráfica
plt.title("Notas Promedio de Cursos")

# Se dan los nombres a los ejes
plt.xlabel("Cursos")
plt.ylabel("Notas")

# Los cursos corresponden a etiquetas y señalamos en qué posición va cada una de␣
 ↪ellas
plt.xticks(posiciones, cursos)

# Se solicita que se muestre por pantalla
plt.show()
```



4.2. Gráficos de Barras Horizontales En este tipo de gráficas, se utiliza la siguiente distribución: * En el eje vertical (eje de las ordenadas), se colocan las variables o grupos de estudio. * En el eje horizontal (eje de las abscisas) se distribuyen los valores que representan la frecuencia de las variables o grupos de estudio.

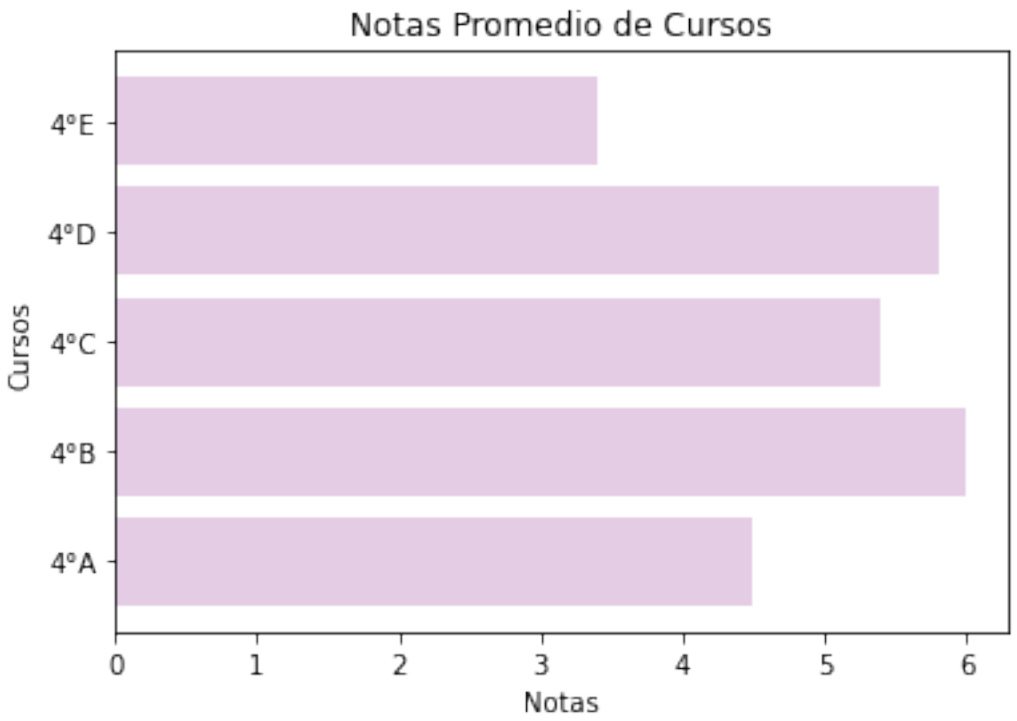
Cabe señalar que todas las barras deben tener el mismo ancho.

Al igual que en la gráfica anterior, Python ofrece la posibilidad de realizar barras horizontales a través de la biblioteca matplotlib mediante la función `.barh()`.

Tanto las características como la forma en realizar este gráfico es idéntica a la generación de gráficos de barras verticales, con excepción del nombre de la función mencionada en el párrafo anterior y que el valor de las barras corresponde a su "ancho" (`width`).

Cabe señalar que hay que tener en consideración que los nombres de las etiquetas deben cambiar, puesto que lo que antes pertenecía al eje x, ahora pertenece al eje y.

[38]:
```
# Ejemplo_15: Se graficarán barras veticales con datos de notas de un curso en una asignatura dada
# Importaciones
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las notas promedio de 5 cursos en una asignatura en particular (eje y)
notas = [4.5, 6.0, 5.4, 5.8, 3.4]

# Se definen las posiciones en el eje x
posiciones = np.arange(len(notas))

# Se definen los cursos (eje x)
cursos = ["4°A", "4°B", "4°C", "4°D", "4°E"]

# SALIDA
# Se genera el gráfico
grafico_barra_vertical = plt.barh(posiciones, notas, color="purple", alpha=0.2)

# Se da nombre a la gráfica
plt.title("Notas Promedio de Cursos")

# Se dan los nombres a los ejes. Nótese el cambio en el orden
plt.xlabel("Notas")
plt.ylabel("Cursos")

# Los cursos corresponden a etiquetas y señalamos en qué posición va cada una de ellas
# Nótese que cambiamos las etiquetas en el eje y, por lo que usamos yticks, en vez de xticks
plt.yticks(posiciones, cursos)

# Se solicita que se muestre por pantalla
plt.show()
```



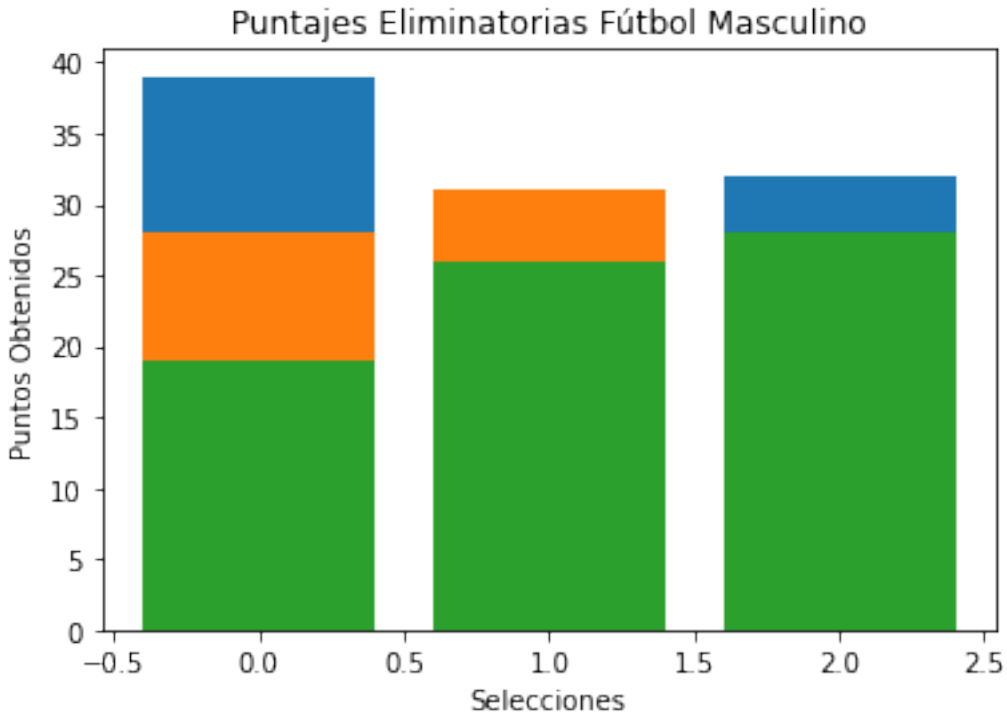
1.3.5 5. Gráficos de Multiseries

Los gráficos de multiseries, también llamados gráficos de datos agrupados, se utilizan para realizar comparaciones entre dos o más conjuntos de datos.

Sus características principales son: * Cada serie de datos se respresenta con el mismo color * Las barras a comparar se colocan una al lado de la otra para facilitar el proceso de análisis.

[39]:
```
# Ejemplo_16: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))
```

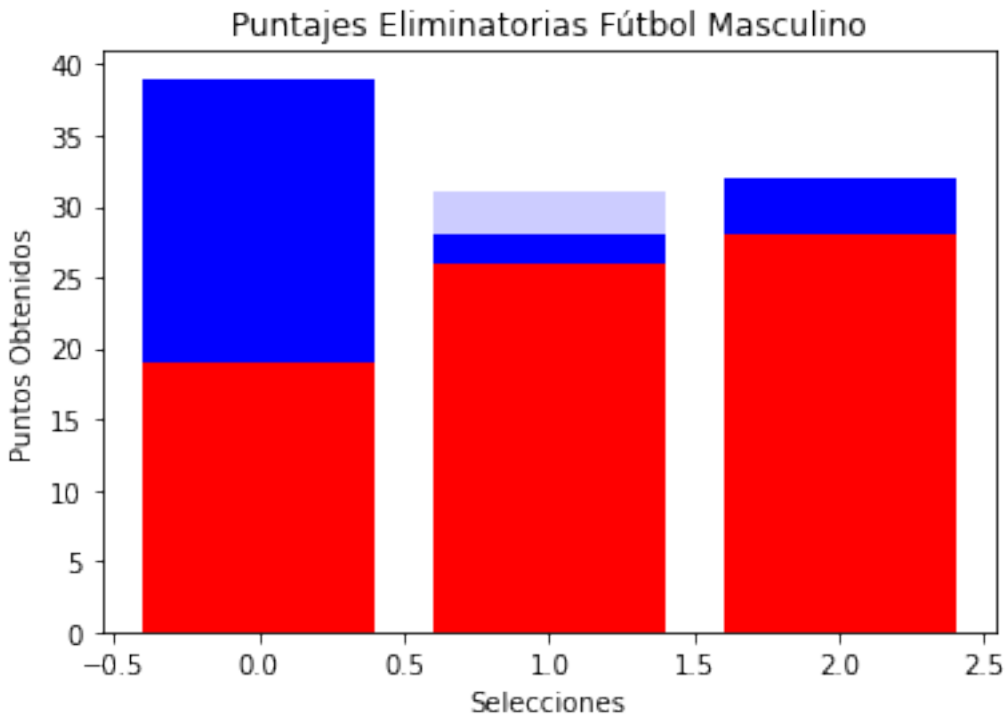
```
# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina)
grafico_barra_urguay = plt.bar(posiciones, uruguay)
grafico_barra_chile = plt.bar(posiciones, chile)

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Se solicita que se muestre por pantalla
plt.show()
```



Como se puede apreciar, en el **Ejemplo_16** no se distingue que barra corresponde a cada selección de fútbol. Para solucionar lo mencionado, se utilizará la característica `**color**`, la cual, como su nombre indica, hará cambiar el color a la barra correspondiente.

[40]:
```
# Ejemplo_17: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
```

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

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, color="b")
grafico_barra_urguay = plt.bar(posiciones, uruguay, color="b", alpha=0.2)
grafico_barra_chile = plt.bar(posiciones, chile, color="r")

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Se solicita que se muestre por pantalla
plt.show()
```

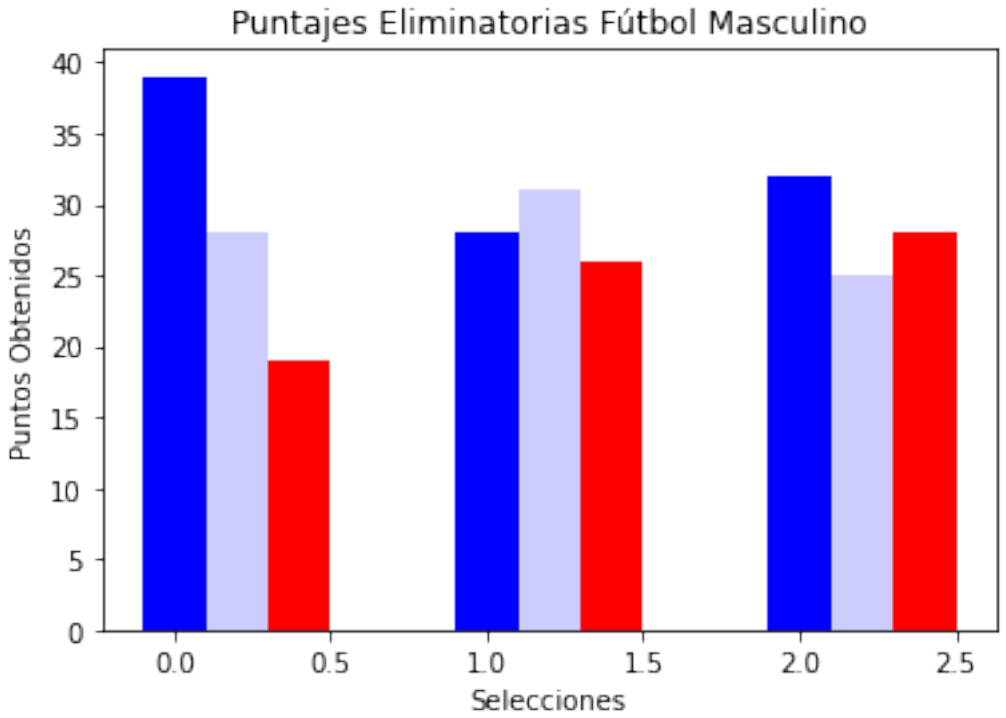


Ahora que se pude distinguir a quien corresponde cada barra, se debe buscar separar las barras para que ninguna de anteponga sobre otra. Para ello, se indicará el ancho de cada barra, buscando que cada una quede al lado de la otra.

Lo anterior se traduce en lo siguiente: * Se dará un ancho de barra de 0.2. * Se indicará que la segunda barra comenzará posterior al finalizar la primera barra, en otras palabra, a la posición de la barra 2, se le suma el ancho (posición + ancho) * Para la posición de la tercera barr,a se hará algo similar, iniciará posterior a dos anchos de barras (posición + 2*ancho)

[42]:
```
# Ejemplo_18: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))
```

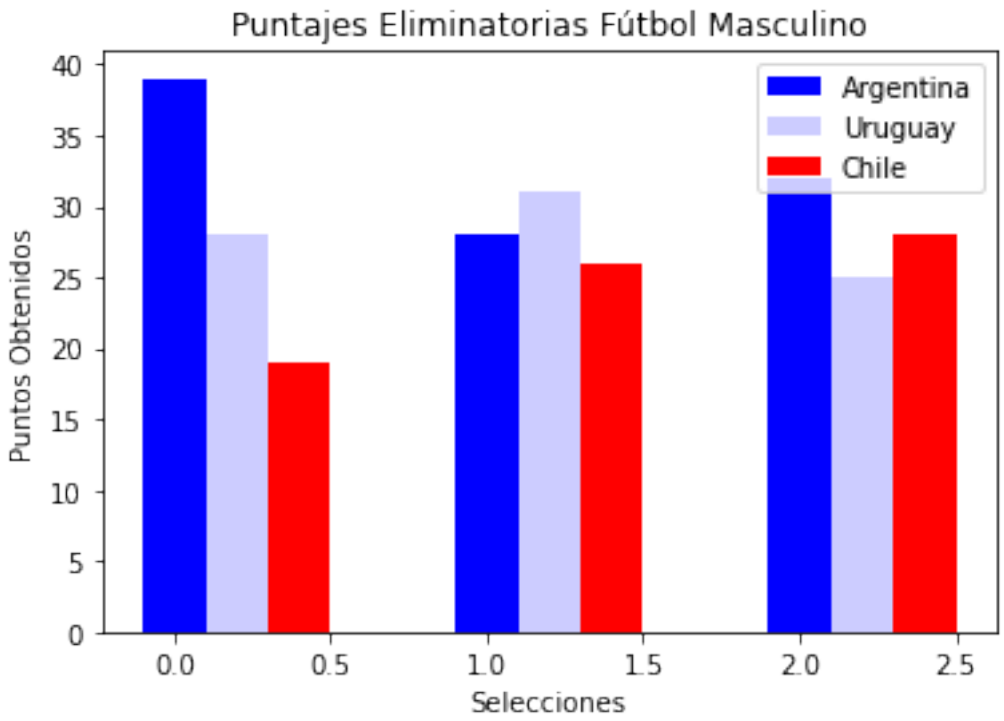
```
# Ancho de barra
ancho = 0.2

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, width=ancho, color="b")
grafico_barra_urguay = plt.bar(posiciones + ancho, uruguay, width=ancho,␣
 ↪color="b", alpha=0.2)
grafico_barra_chile = plt.bar(posiciones + 2*ancho, chile, width=ancho,␣
 ↪color="r")

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Se solicita que se muestre por pantalla
plt.show()
```



En el **Ejemplo_18** ya se lograron crear las 3 barras de las variables a analizar, pero ¿qué barra representa a Chile, a Argetina y a Uruguay?

Para distinguirlas, se utilizará la opción de colocar leyenda (`label` en la creación de cada barra y el método `.legend()` en el Bloque Principal)

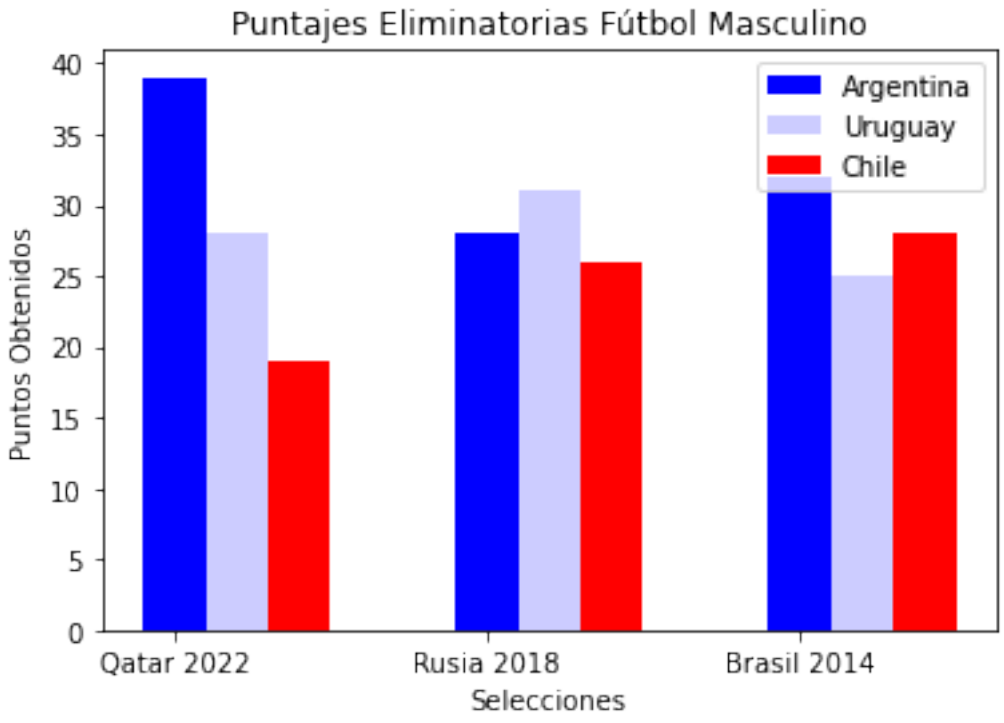
[45]:
```
# Ejemplo_19: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# Ancho de barra
ancho = 0.2

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, width=ancho,
                                  color="b", label="Argentina")
grafico_barra_uruguay = plt.bar(posiciones + ancho, uruguay, width=ancho,
                                color="b", alpha=0.2, label="Uruguay")
grafico_barra_chile = plt.bar(posiciones + 2*ancho, chile, width=ancho,
                              color="r", label="Chile")

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Añade leyenda
plt.legend()

# Se solicita que se muestre por pantalla
plt.show()
```



Lo último que falta para que cualquier persona pueda entender lo que se quiere mostrar en el gráfico es agregarle los nombres de cada mundial en cada eliminatoria que se está comparando. Para ello, se indicarán las etiquetas y se reemplazarán por las posiciones en el eje x.

[46]:
```
# Ejemplo_20: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
#Se definen las etiquetas correspondientes a los mundiales (eje x)
mundiales = ["Qatar 2022", "Rusia 2018", "Brasil 2014"]

# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# Ancho de barra
```

```
ancho = 0.2

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, width=ancho,
                                  color="b", label="Argentina")
grafico_barra_uruguay = plt.bar(posiciones + ancho, uruguay, width=ancho,
                                color="b", alpha=0.2, label="Uruguay")
grafico_barra_chile = plt.bar(posiciones + 2*ancho, chile, width=ancho,
                              color="r", label="Chile")

#Se asignan las etiquetas a las posiciones del eje X
plt.xticks(posiciones, mundiales)

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Añade leyenda
plt.legend()

# Se solicita que se muestre por pantalla
plt.show()
```



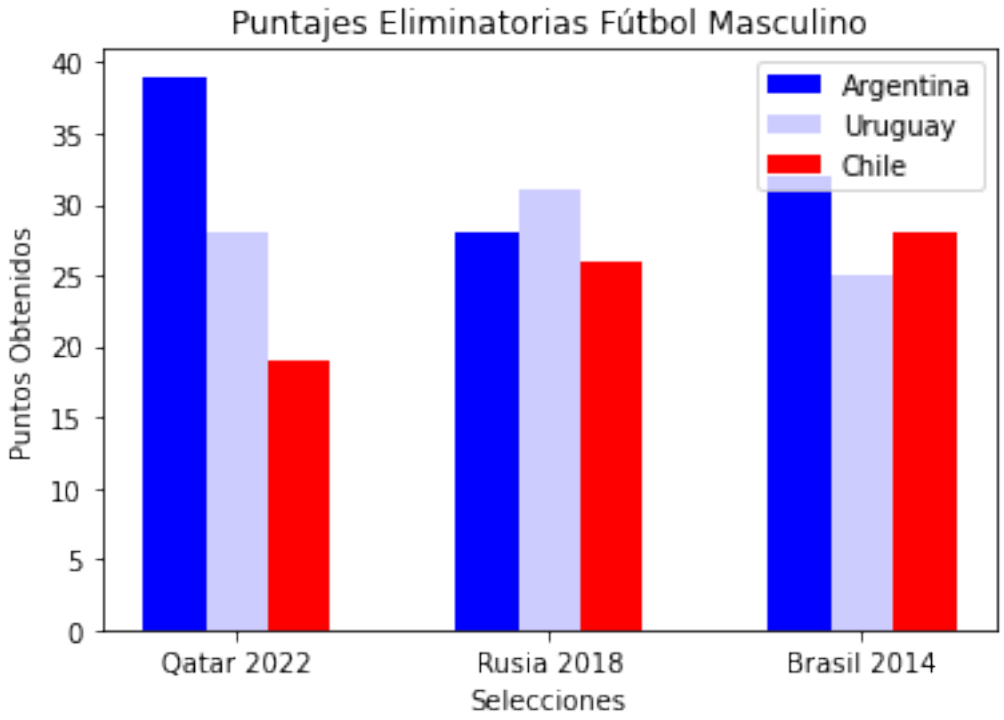
El nombre de las etiquetas queda algo inclinado hacia un lado. Para poder mejorar visualmente la presentación se cambiará el lugar en donde se mostrará la etiqueta correspondiente a cada mundial. Esto se llevará a cabo dentro del método `.xticks()` de la siguiente manera:

```
.xticks(etiquetas + ancho, mundiales)
```

De esa manera, la etiqueta quedará desplazada un valor equivalente al ancho.

[47]:
```
# Ejemplo_21: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
#Se definen las etiquetas correspondientes a los mundiales (eje x)
mundiales = ["Qatar 2022", "Rusia 2018", "Brasil 2014"]

# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# Ancho de barra
ancho = 0.2

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, width=ancho,
                                  color="b", label="Argentina")
grafico_barra_uruguay = plt.bar(posiciones + ancho, uruguay, width=ancho,
                                color="b", alpha=0.2, label="Uruguay")
grafico_barra_chile = plt.bar(posiciones + 2*ancho, chile, width=ancho,
                              color="r", label="Chile")

#Se asignan las etiquetas a las posiciones del eje X
plt.xticks(posiciones + ancho, mundiales)

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
```

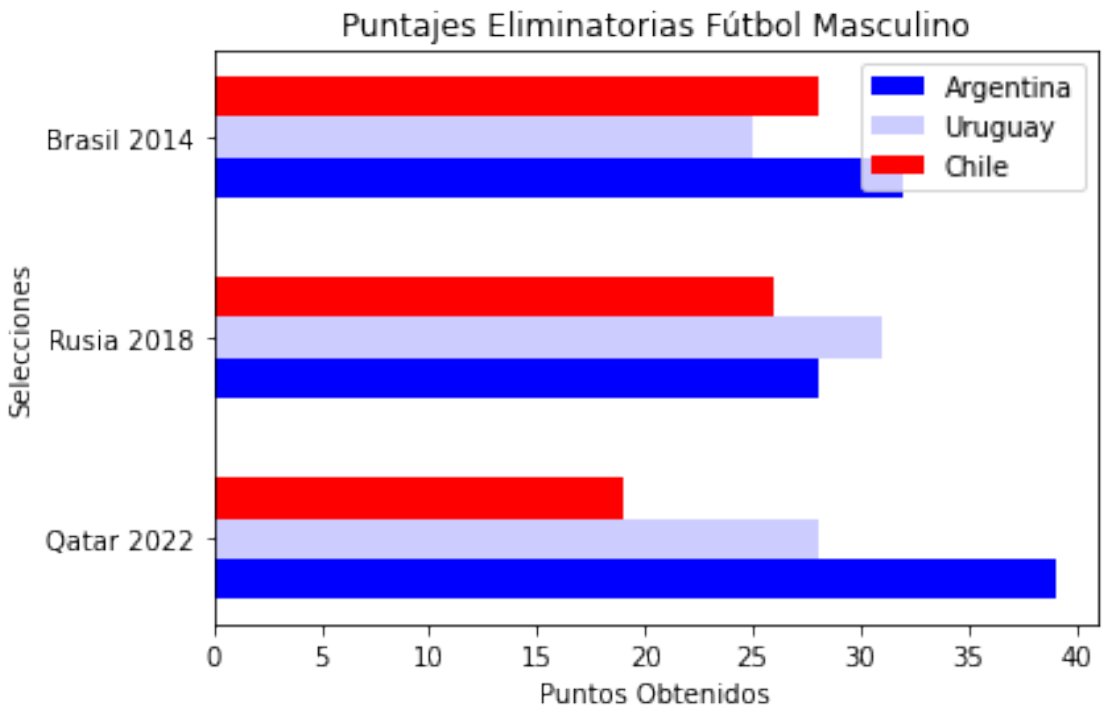
```
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Añade leyenda
plt.legend()

# Se solicita que se muestre por pantalla
plt.show()
```



¿Cómo podríamos hacerlo si queremos un gráfico de barras horizontales? Sencillamente cambiamos las referencias al eje x por el eje y y, en la función `barh`, especificamos que el *alto* de las barras será 0.2, en lugar del ancho (y propagando el cambio a todos los lugares donde haga falta):

[48]:
```
# Ejemplo_22: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
#Se definen las etiquetas correspondientes a los mundiales (eje x)
mundiales = ["Qatar 2022", "Rusia 2018", "Brasil 2014"]
```

```
# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# Ancho de barra
alto = 0.2

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.barh(posiciones, argentina, height=alto,
                                   color="b", label="Argentina")
grafico_barra_uruguay = plt.barh(posiciones + alto, uruguay, height=ancho,
                                 color="b", alpha=0.2, label="Uruguay")
grafico_barra_chile = plt.barh(posiciones + 2*alto, chile, height=ancho,
                               color="r", label="Chile")

#Se asignan las etiquetas a las posiciones del eje X
plt.yticks(posiciones + alto, mundiales)

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Puntos Obtenidos")
plt.ylabel("Selecciones")

# Añade leyenda
plt.legend()

# Se solicita que se muestre por pantalla
plt.show()
```



1.3.6 6. Múltiples Gráficos

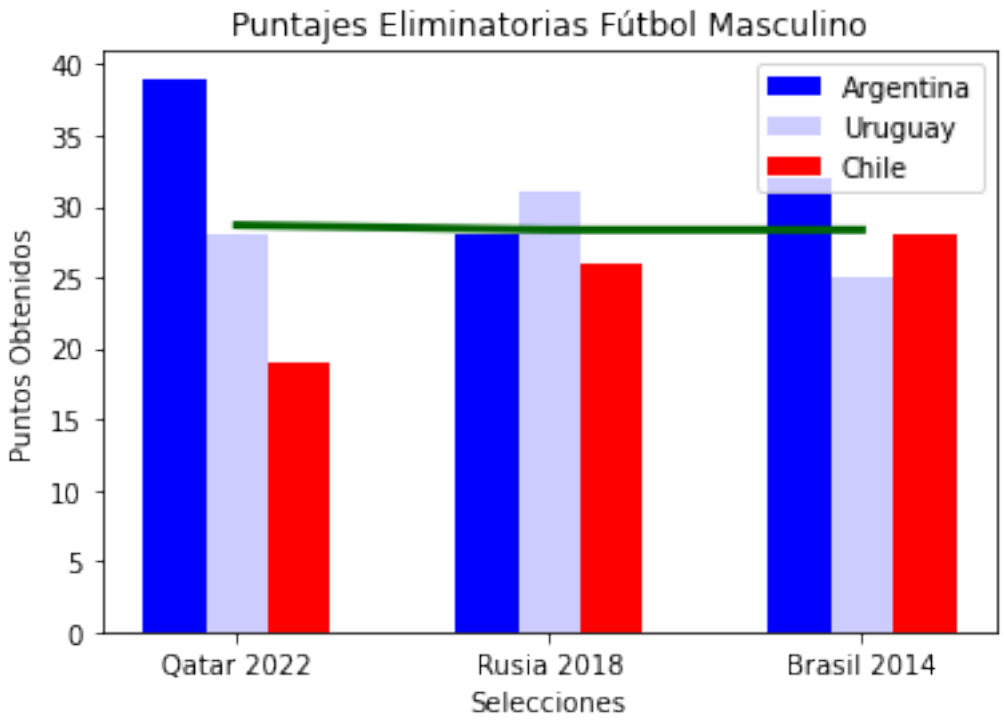
En ocasiones, la realización de un solo gráfico no es suficiente para poder tomar decisiones. A modo de ejemplo, para poder analizar los problemas que una empresa tiene, se utiliza el diagrama de Pareto, el cual utiliza de manera simultánea gráficos de barra con una línea recta.

Para ejemplificar lo anterior, se agregará una línea recta al gráfico de barras realizada en el **Ejemplo_21**.

Para ello, se debe definir la curva a graficar y generar la recta con ayuda del método **.plot()**

[49]:
```
# Ejemplo_23: Se graficarán barras de múltiples series respecto a puntajes␣
 ↪obtenidos en las
# últimas eliminatorias Sudamericanas de Fútbol masculino
import matplotlib.pyplot as plt
import numpy as np

# BLOQUE PRINCIPAL
# Procesamiento
# Se definen las etiquetas correspondientes a los mundiales (eje x)
mundiales = ["Qatar 2022", "Rusia 2018", "Brasil 2014"]

# Se definen los puntajes de las selecciones en estudio
argentina = [39, 28, 32]
uruguay = [28, 31, 25]
```

```
chile = [19, 26, 28]

# Se definen las posiciones que utilizará cada mundial en el eje X
posiciones = np.arange(len(argentina))

# Ancho de barra
ancho = 0.2

#Se generan los valores del eje Y mediante el promedio de los puntajes obtenidos␣
 ↪de las selecciones en estudio
promedio_puntos = (np.array(argentina) + np.array(uruguay) + np.array(chile))/3

# SALIDA
# Se genera el gráfico
grafico_barra_argentina = plt.bar(posiciones, argentina, width=ancho,
                                  color="b", label="Argentina")
grafico_barra_uruguay = plt.bar(posiciones + ancho, uruguay, width=ancho,
                                color="b", alpha=0.2, label="Uruguay")
grafico_barra_chile = plt.bar(posiciones + 2*ancho, chile, width=ancho,
                              color="r", label="Chile")


recta_promedios = plt.plot(posiciones+ancho, promedio_puntos, color="darkgreen",␣
 ↪linewidth=3)

# Se asignan las etiquetas a las posiciones del eje X
plt.xticks(posiciones + ancho, mundiales)

# Se da nombre a la gráfica
plt.title("Puntajes Eliminatorias Fútbol Masculino")

 #Se dan los nombres a los ejes
plt.xlabel("Selecciones")
plt.ylabel("Puntos Obtenidos")

# Añade leyenda
plt.legend()

# Se solicita que se muestre por pantalla
plt.show()
```



1.4 Múltiples Gráficos en la misma Figura

Los ejemplos anteriores muestran distintas gráficas que se pueden crear en Python. No obstante, cada una de ellas se realizó en figuras independientas unas de otras.

Python ofrece la opción de crear gráficas distintas en la misma figura. Para ello, la figura o pantalla se divide en determiandas partes y distribuye las gráficas definidas en las posiciones que el usuario determine.

Para lograr lo anterior, la biblioteca `matplotlib` ofrece el método `.subplot()`, el cual recibe 3 parámetros:

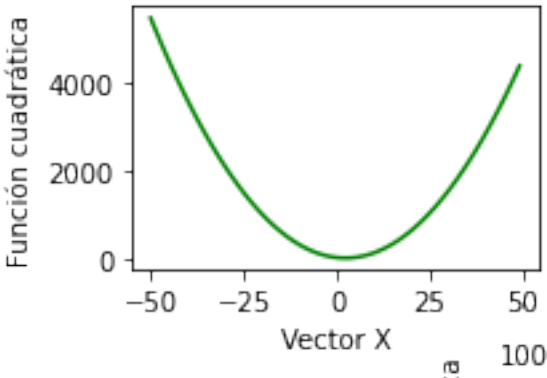
`subplot(x, y, z)`

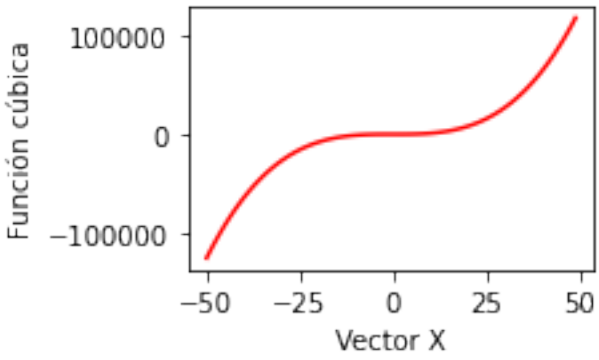
- `x`: la cantidad de filas en que se dividirá la figura
- `y`: la cantidad de filas en que se dividirá la figura
- `z`: la ubicación en donde se formará la gráfica deseada. Corresponde al número de figura, contando desde 1 hasta `x*y`, de izquierda a derecha y de arriba hacia abajo.

En el **Ejemplo_23** se creará una figura con dos gráficas, la primera ubicada en la posición 1 y la otra, en la posición 4.

[51]:
```
# Ejemplo_21: Se graficará 2 curvas utilizando la función plot() con 2 parámetros
import numpy as np
import matplotlib.pyplot as plt
```

```
# BLOQUE PRINCIPAL
# Procesamiento
# Se genera el vector que será utilizado como base para obtener el eje y
vector = np.arange(-50, 50)

# Se genera el eje y
funcion_1 = 2*vector**2 - 9*vector+10
funcion_2 = vector**3 - 2*vector - 3

# Salida
# En el apartado salida se genera el gráfico
# Se indica que se crearán 2 filas y 1 columna. La gráfica siguiente se ubicará␣
 ↪en la 1° posición
plt.subplot(2, 2, 1)

# Se generan el gráfico _1
grafico_1 = plt.plot(vector, funcion_1)
plt.setp(grafico_1, "color", "green")

# Se dan los nombres a los ejes
plt.xlabel("Vector X")
plt.ylabel("Función cuadrática")

# Se indica que se crearán 2 filas y 1 columna. La gráfica siguiente se ubicará␣
 ↪en la 2° posición
plt.subplot(2, 2, 4)

# Se generan el gráfico_2
grafico_2 = plt.plot(vector, funcion_2)
plt.setp(grafico_2, "color", "red")

# Se dan los nombres a los ejes
plt.xlabel("Vector X")
plt.ylabel("Función cúbica")

# Se solicita que se muestre por pantalla
plt.show()
```





2 Bibliografía

J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

Sundnes, J. (2020). Arrays and Plotting. En Introduction to Scientific Programming with Python (1.a ed., p. 81). Springer. https://doi.org/10.1007/978-3-030-50356-7

2.1 Enlaces útiles:

- Tutoriales: https://matplotlib.org/stable/tutorials/index.html
- Expresiones matemáticas: https://matplotlib.org/stable/tutorials/text/mathtext.html
- Marcadores: https://matplotlib.org/stable/api/markers_api.html
- Colores: https://matplotlib.org/3.5.0/tutorials/colors/colors.html
- Documentación completa de `plot`: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.htm using-matplotlib-pyplot-plot