Python Fundamentals

June 24, 2022

1 INTRODUCCIÓN A PYTHON

1.1 BÁSICOS

type(df) permite conocer el tipo de una variable (float -números reales-, integer, string, boolean)

str() convierte variables a strings, float() a números

; se usa para poner diferentes comandos en la misma línea

1.2 LISTAS

Las listas se construyen con corchetes []

[843]:
```
list1 = [5, 9, 5, 2]

type(list1)
```

[843]:
```
list
```

1.2.1 SUBCONJUNTOS DE LISTAS

[844]:
```
# Recuérdese que las listas o vectores comienzan con el elemento 0

print(list1[3])

# También se pueden usar índices negativos:

print(list1[-1])

# Slicing (donde se incluye e inicio, pero no el final)

print(list1[1:3])

# El slicing comenzará desde 0 si se escribe como sigue:

print(list1[:3])

# E incluirá el último elemento si se escribe:

```

```
print(list1[2:])
```

```
2
2
[9, 5]
[5, 9, 5]
[5, 2]
```

[845]:
```
# Ejemplo

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.
 ↪75, "bathroom", 9.50]

print(areas[1])

eat_sleep_area = (areas[3] + areas[7])
print(eat_sleep_area)

downstairs = areas[:6]
upstairs = areas[6:10]
print(downstairs)
print(upstairs)
```

```
11.25
28.75
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]
['bedroom', 10.75, 'bathroom', 9.5]
```

[846]:
```
# Manipulacio´n de listas

# Para actualizar un elemento de una lista:

list1[2] = 6
print(list1)

list1[0:2] = [9, 10]
print(list1)

# Al sumar listas, se pegan:

print(list1 + [55, 21])

# Y para eliminar elementos de una lista, se usa del():

del(list1[1])
print(list1)
```

```
[5, 9, 6, 2]
[9, 10, 6, 2]
```

```
[9, 10, 6, 2, 55, 21]
[9, 6, 2]
```

[847]:
```
# Para evitar que una copia de una lista se vea afectada por cambios en la␣
 ↪original:

areas_copy = list(areas)

areas_copy[1] = 4

print(areas)
print(areas_copy)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75,
'bathroom', 9.5]
['hallway', 4, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75,
'bathroom', 9.5]
```

1.3 FUNCIONES Y PAQUETES

[848]:
```
print(max(list1))

print(round(areas[1], 1))
print(round(areas[9]))
print(len(areas)) #lenght

# La función help() se usa para obtener ayuda de las funciones:

help(round)

###

first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]
full = first + second
full_sorted = sorted(full, reverse = True)
print(full_sorted)

# Los métodos normalmente se escriben después de un punto, por ejemplo, para␣
 ↪indexar o contar:

print(full_sorted.count(18.0))
print(areas.index("kitchen"))

list5 = "hola"
print(list5.capitalize())
print(list5.replace("h", "H"))
full_sorted.append(65)
```

```
full_sorted
```

```
9
11.2
10
10
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.

[20.0, 18.0, 11.25, 10.75, 9.5]
1
2
Hola
Hola
```

[848]: [20.0, 18.0, 11.25, 10.75, 9.5, 65]

1.4 NUMPY

[849]:
```
# La matriz de NumPy tiene permite realizar cálculos en listas completas

import numpy as np

height = [1.73, 1.68, 1.71, 1.89, 1.79]
np_height = np.array(height)
print(np_height)

weight = [65.4, 59.2, 63.6, 88.4, 68.7]
np_weight = np.array(weight)
print(np_weight)

# Entonces:

bmi = np_weight / np_height ** 2
print(bmi)

# Nótese que esto es posible porque NumPy asume que todos los elementos de una␣
 ↪lista son del mismo tipo y, por lo tanto, puede
# realizarse una operación con ellos

# Para obtener subconjuntos de array de NumPy:

print(bmi[1])
```

```
print(bmi > 23)

print(bmi[bmi > 23])
```

```
[1.73 1.68 1.71 1.89 1.79]
[65.4 59.2 63.6 88.4 68.7]
[21.85171573 20.97505669 21.75028214 24.7473475  21.44127836]
20.97505668934241
[False False False  True False]
[24.7473475]
```

1.4.1 2D NumPy arrays

[850]:
```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, 68.7]])

print(np_2d)

print(np_2d.shape)

print(np_2d[0, 2])

print(np_2d[:, 1:3])
```

```
[[ 1.73  1.68  1.71  1.89  1.79]
 [65.4  59.2  63.6  88.4  68.7 ]]
(2, 5)
1.71
[[ 1.68  1.71]
 [59.2  63.6 ]]
```

[851]:
```
# Ejemplo baseball

baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]

np_baseball = np.array(baseball)

print(type(np_baseball))
print(np_baseball.shape)

print(np_baseball[2,:])
np_weight_lb = np_baseball[:,1]
print(np_baseball[3, 0])
```

```
<class 'numpy.ndarray'>
```

```
(4, 2)
[210.   98.5]
188.0
```

1.4.2 Estadísticas básicas NumPy

[852]:
```
# np.mean(df[:,])

# np.median(df[:,])

# np.corrcoef(df[,], df[,])

# np.std(df[,])

# np.sum(df[,])

#np.sort(df[,])

# ---> La mayor ventaja de calcular summary statistics con NumPy es la␣
 ↪velocidad con respecto de las funciones básicas de Pyhton

# También pueden simularse datos:

height = np.round(np.random.normal(1.75, 0.20, 5000), 2)
weight = np.round(np.random.normal(60, 15, 5000), 2)

np_city = np.column_stack((height, weight))

print(np_city)
```

```
[[ 1.56 62.44]
 [ 1.65 70.07]
 [ 1.96 64.3 ]
 …
 [ 1.87 33.8 ]
 [ 2.06 38.88]
 [ 2.29 34.89]]
```

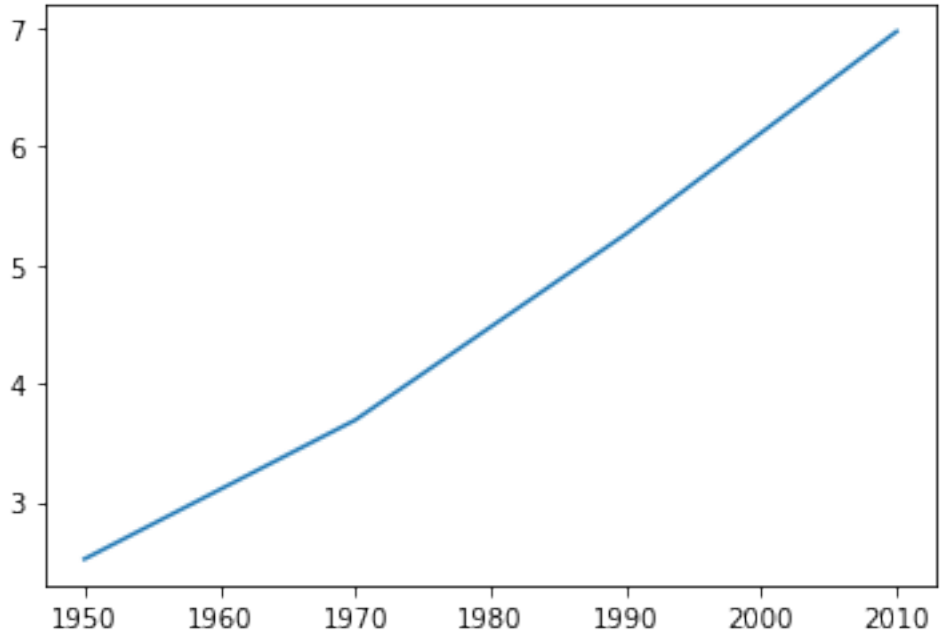
2 PYTHON INTERMEDIO

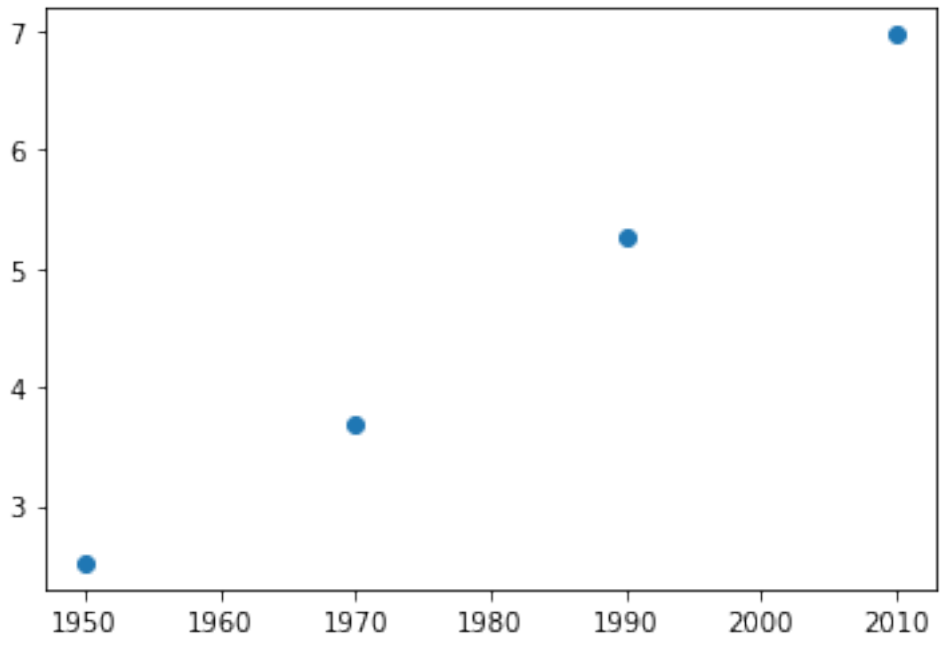
2.1 MATPLOTLIB

2.1.1 Gráficas Básicas

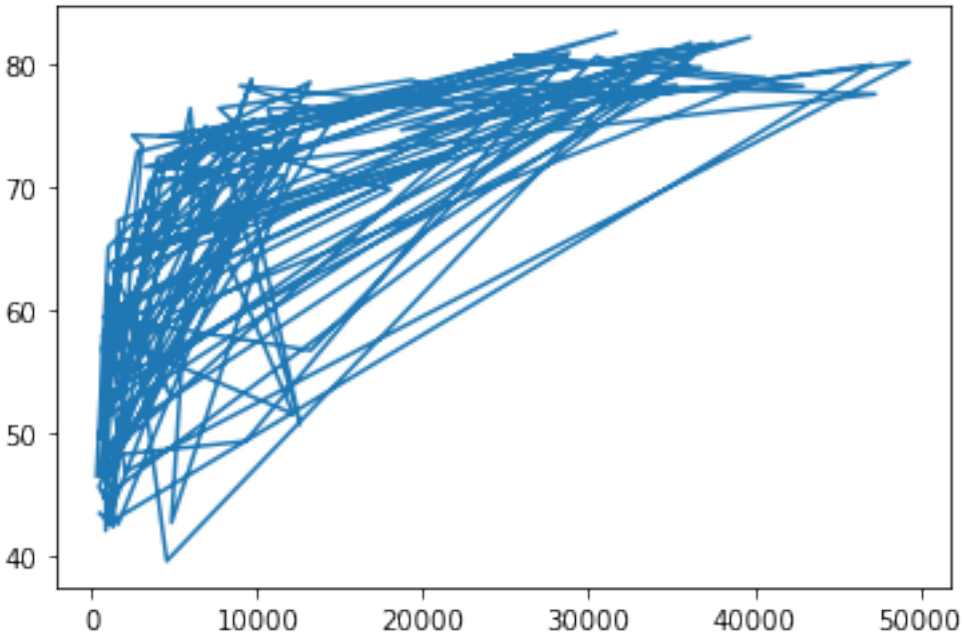
[853]:
```
import matplotlib.pyplot as plt

year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
```

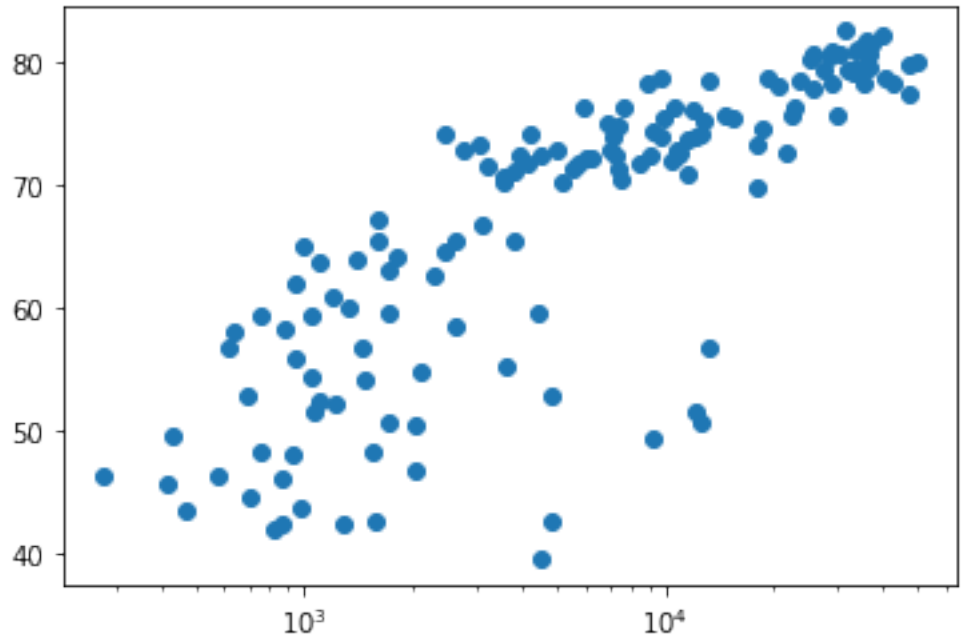
```
plt.plot(year, pop)
plt.show()

plt.scatter(year, pop)
plt.show()

# Para transformar un eje en log:

# plt.xscale("log")
```

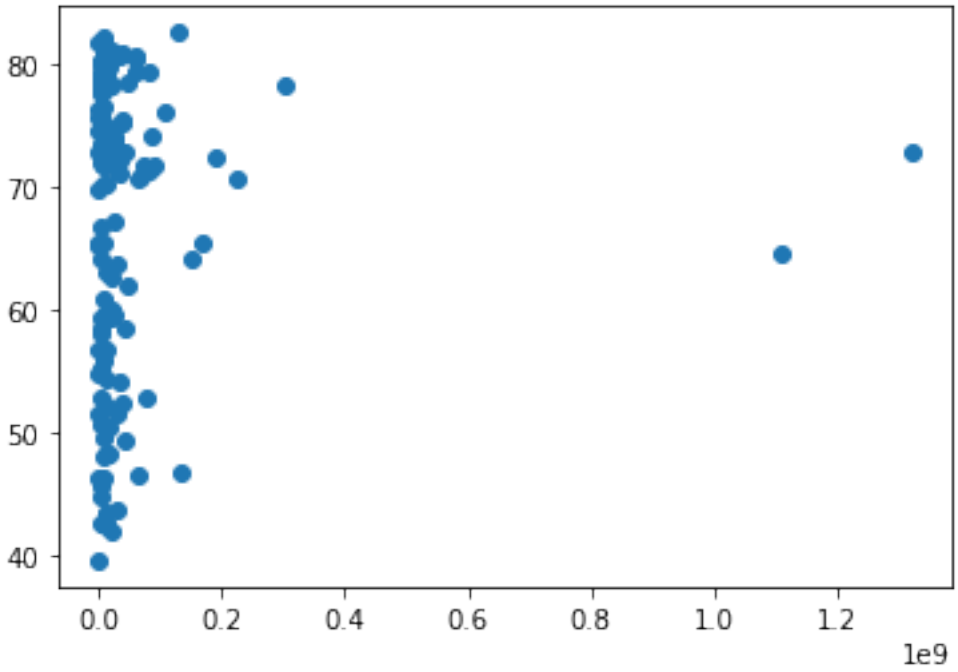




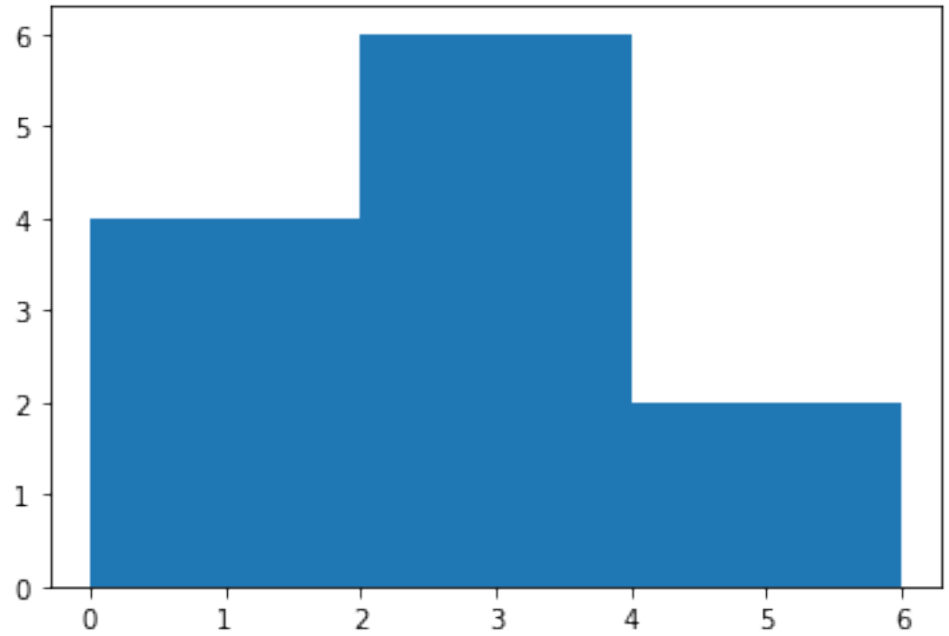
[854]:
```
# Ejemplo Gapminder

import matplotlib.pyplot as plt
import pandas as pd

gapminder = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/gapminder.
 ↪csv")

plt.plot(gapminder["gdp_cap"], gapminder["life_exp"])
plt.show()

# Es necesario hacer un scatter para una mejor visualización

plt.scatter(gapminder["gdp_cap"], gapminder["life_exp"])
plt.xscale("log")
plt.show()

###

plt.scatter(gapminder["population"], gapminder["life_exp"]); plt.show()
```


80
70
60
50
40
0
10000
20000
30000
40000
50000

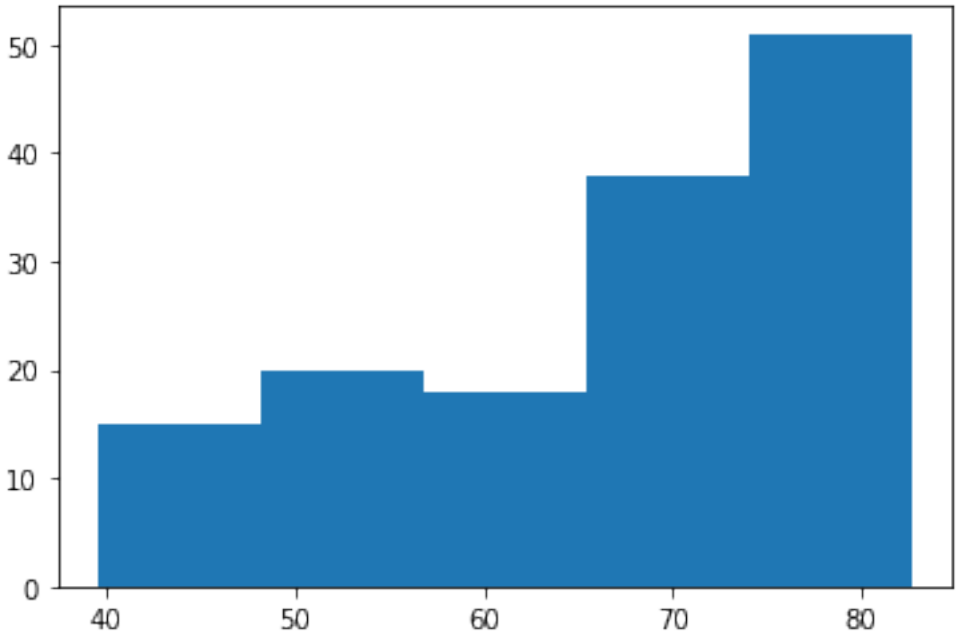

80
70
60
50
40
10^3
10^4

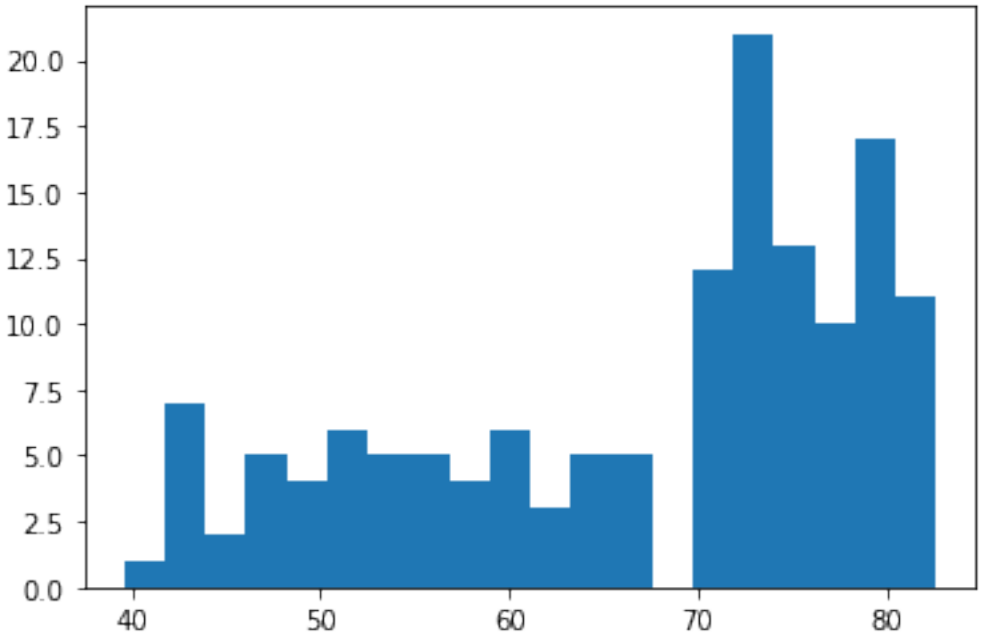


[855]:
```
# Histogramas

values = [0, 0.6, 1.4, 1.6, 2.2, 2.5, 2.6, 3.2, 3.5, 3.9, 4.2, 6]
plt.hist(values, bins = 3)
plt.show()
```



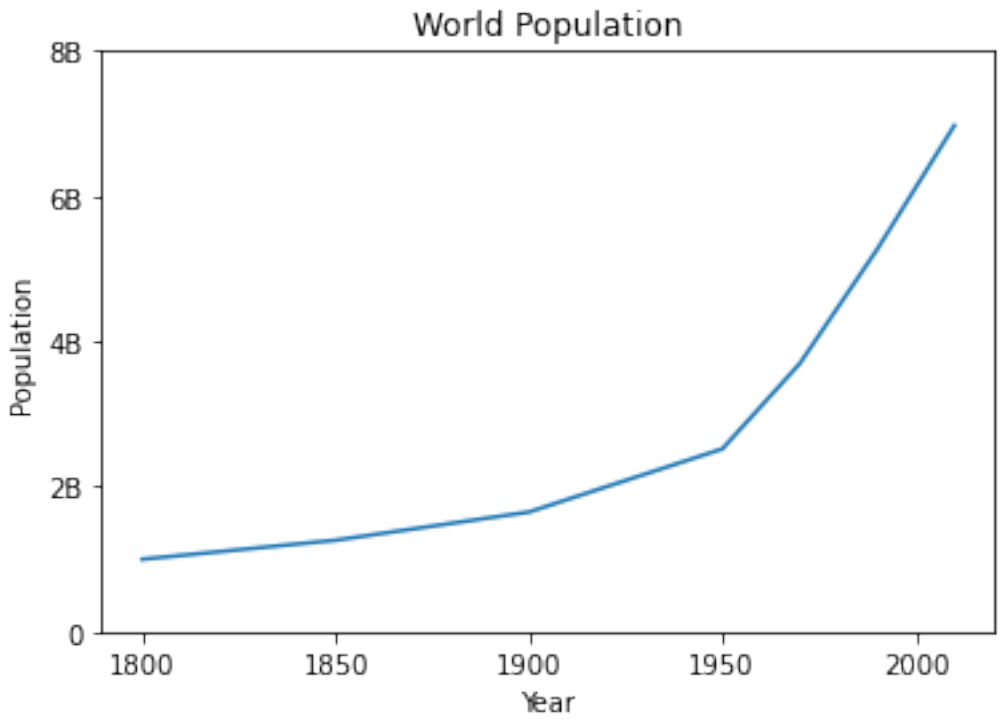
[856]:
```
# Ejemplo Gapminder

plt.hist(gapminder["life_exp"], bins = 5)
plt.show()

plt.hist(gapminder["life_exp"], bins = 20)
plt.show()
```





2.1.2 Personalización

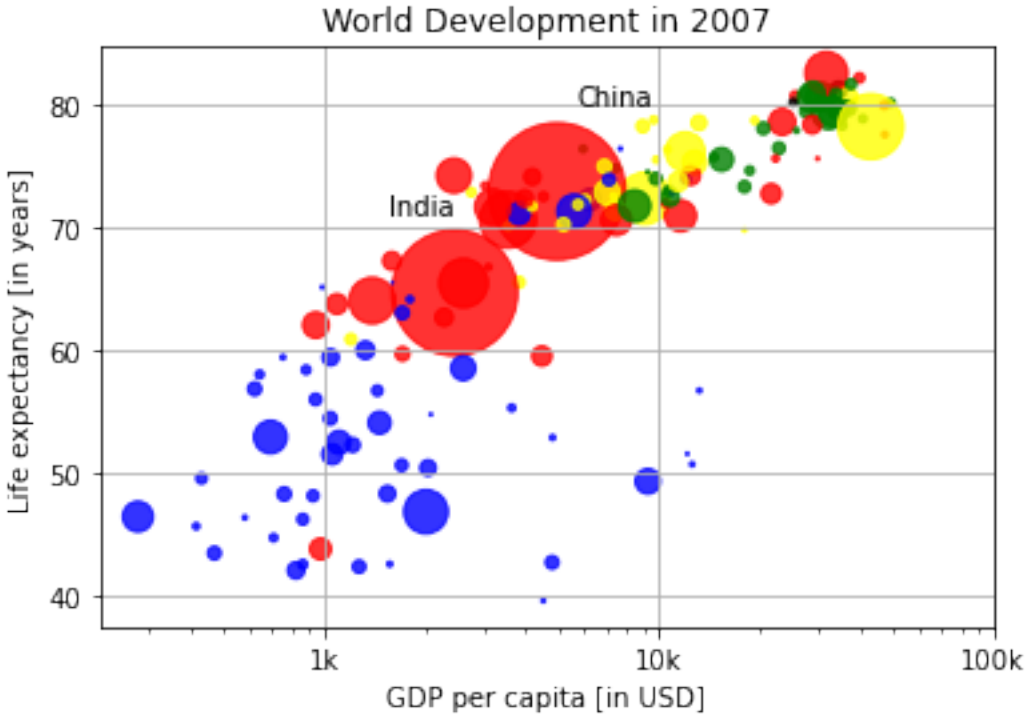
[857]:
```
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)
plt.xlabel("Year")
plt.ylabel("Population")
plt.title("World Population")
plt.yticks([0, 2, 4, 6, 8], ["0", "2B", "4B", "6B", "8B"])
plt.show()
```



[858]:
```
# Ejemplo Gapminder

import numpy as np
```

```
pop = [31.889923, 3.600523, 33.333216, 12.420476, 40.301927, 20.434176, 8.
 ↪199783, 0.708573, 150.448339, 10.392226, 8.078314, 9.119152, 4.552198, 1.
 ↪639131, 190.010647, 7.322858, 14.326203, 8.390505, 14.131858, 17.696293, 33.
 ↪390141, 4.369038, 10.238807, 16.284741, 1318.683096, 44.22755, 0.71096, 64.
 ↪606759, 3.80061, 4.133884, 18.013409, 4.493312, 11.416987, 10.228744, 5.
 ↪46812, 0.496374, 9.319622, 13.75568, 80.264543, 6.939688, 0.551201, 4.
 ↪906585, 76.511887, 5.23846, 61.083916, 1.454867, 1.688359, 82.400996, 22.
 ↪873338, 10.70629, 12.572928, 9.947814, 1.472041, 8.502814, 7.483763, 6.
 ↪980412, 9.956108, 0.301931, 1110.396331, 223.547, 69.45357, 27.499638, 4.
 ↪109086, 6.426679, 58.147733, 2.780132, 127.467972, 6.053193, 35.610177, 23.
 ↪301725, 49.04479, 2.505559, 3.921278, 2.012649, 3.193942, 6.036914, 19.
 ↪167654, 13.327079, 24.821286, 12.031795, 3.270065, 1.250882, 108.700891, 2.
 ↪874127, 0.684736, 33.757175, 19.951656, 47.76198, 2.05508, 28.90179, 16.
 ↪570613, 4.115771, 5.675356, 12.894865, 135.031164, 4.627926, 3.204897, 169.
 ↪270617, 3.242173, 6.667147, 28.674757, 91.077287, 38.518241, 10.642836, 3.
 ↪942491, 0.798094, 22.276056, 8.860588, 0.199579, 27.601038, 12.267493, 10.
 ↪150265, 6.144562, 4.553009, 5.447502, 2.009245, 9.118773, 43.997828, 40.
 ↪448191, 20.378239, 42.292929, 1.133066, 9.031088, 7.554661, 19.314747, 23.
 ↪174294, 38.13964, 65.068149, 5.701579, 1.056608, 10.276158, 71.158647, 29.
 ↪170398, 60.776238, 301.139947, 3.447496, 26.084662, 85.262356, 4.018332, 22.
 ↪211743, 11.746035, 12.311143]

colors = {
    'Asia':'red',
    'Europe':'green',
    'Africa':'blue',
    'Americas':'yellow',
    'Oceania':'black'
}

np_pop = np.array(pop)
np_pop = np_pop*2

plt.scatter(x = gapminder["gdp_cap"], y = gapminder["life_exp"], s = np_pop,␣
 ↪alpha = 0.8, c = gapminder["cont"].map(colors))
plt.xscale("log")
plt.xlabel("GDP per capita [in USD]")
plt.ylabel("Life expectancy [in years]")
plt.title("World Development in 2007")

tick_val = [1000, 10000, 100000]
tick_lab = ["1k", "10k", "100k"]
plt.xticks(tick_val, tick_lab)

plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')
plt.grid(True)
```

```
plt.show()
```



2.2 DICCIONARIOS & PANDAS

2.2.1 Diccionarios 1

[859]:
```
# Puede usarse el método de indexar:

pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]
ind_alb = countries.index("albania")
print(pop[ind_alb])

# O bien, un diccionario:

world = {
    "afghanistan": 30.55,
    "albania": 2.77,
    "algeria": 39.21
}

print(world["albania"])
```

```
2.77
2.77
```

[860]:
```
# Ejemplo

countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

ind_ger = countries.index("germany")

print(capitals[ind_ger])

###

countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

europe = {
    "spain": "madrid",
    "france": "paris",
    "germany": "berlin",
    "norway": "oslo"
}

print(europe)

print(europe.keys())
```

```
berlin
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo'}
dict_keys(['spain', 'france', 'germany', 'norway'])
```

2.2.2 Diccionarios 2

[861]:
```
# Para adicionar más datos a un diccionario existente:

world["sealand"] = 0.000028

print(world)

# Si se tiene una colección de valores, en donde el orden importa y se quiere␣
 ↪seleccionar subconjuntos enteros,
# es preferible utilizar una lista

# Si en cambio se requiere inspeccionar y localizar datos de un conjunto, es␣
 ↪preferible usar diccionarios
```

```
{'afghanistan': 30.55, 'albania': 2.77, 'algeria': 39.21, 'sealand': 2.8e-05}
```

[862]:
```
# Ejemplo países

europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw',
          'australia':'vienna' }

europe["germany"] = "berlin"

del(europe["australia"])

print(europe)

data = {
    "capital": "rome",
    "population": 59.83
}

europe["italy"] = data

print(europe)
```

```
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo',
'italy': 'rome', 'poland': 'warsaw'}
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo',
'italy': {'capital': 'rome', 'population': 59.83}, 'poland': 'warsaw'}
```

2.2.3 Pandas 1

[863]:
```
# Al trabajar con datos de diferentes tipos, las matrices de NumPy no son tan␣
 ↪útiles.
# Para eso, es mejor utilizar Pandas

brics = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/brics.csv",␣
 ↪index_col = 0) # index_col se usa para
# indicar que la primera columna es el índice de renglones
print(brics)
```

```
         country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
```

[864]:
```
# Ejemplo

names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco',␣
 ↪'Egypt']
```

```
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]

import pandas as pd

my_dict = {
    "country": names,
    "drives_right": dr,
    "cars_per_cap": cpc
}

cars = pd.DataFrame(my_dict)
print(cars)

###

row_labels = ['US', 'AUS', 'JPN', 'IN', 'RU', 'MOR', 'EG']

cars.index = row_labels
print(cars)
```

```
         country  drives_right  cars_per_cap
0  United States          True           809
1      Australia         False           731
2          Japan         False           588
3          India         False            18
4         Russia          True           200
5        Morocco          True            70
6          Egypt          True            45
           country  drives_right  cars_per_cap
US   United States          True           809
AUS      Australia         False           731
JPN          Japan         False           588
IN           India         False            18
RU          Russia          True           200
MOR        Morocco          True            70
EG           Egypt          True            45
```

2.2.4 Pandas 2

[865]:
```
# Hay varias formas para indexar y seleccionar data

# Corchetes:

print(brics["country"])
print(type(brics["country"]))

# Al pegar varias listas, se puede crear un dataframe
```

```
# Pero para crear un dataframe de solo esta columna, se usan dobles corchetes:

print(brics[["country", "capital"]])
print(type(brics[["country", "capital"]]))

# Alternativamente se puede utilizar loc y iloc:

print(brics.loc["RU"])

print(brics.loc[["RU"]])

print(brics.loc[["RU", "IN", "CH"]])

print(brics.loc[["RU", "IN", "CH"], ["country", "capital"]])

print(brics.loc[:, ["country", "capital"]])

# Los corchetes funcionan bien para obtener columnas: brics[["country",␣
 ↪"capital"]]
# El slicing funciona para acceder a filas: brics[1:3]
# loc permite acceder a ambas simultáneamente: brics.loc[:, ["country",␣
 ↪"capital"]]

# loc se basa en nombres de etiquetas; por su parte, iloc funciona con base en␣
 ↪la posición o índice

print(brics.iloc[:, [0,1]])
```

```
BR          Brazil
RU          Russia
IN           India
CH           China
SA    South Africa
Name: country, dtype: object
<class 'pandas.core.series.Series'>
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
<class 'pandas.core.frame.DataFrame'>
country       Russia
capital       Moscow
area            17.1
population     143.5
```

```
Name: RU, dtype: object
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
   country    capital    area  population
RU  Russia     Moscow  17.100       143.5
IN   India  New Delhi   3.286      1252.0
CH   China    Beijing   9.597      1357.0
   country    capital
RU  Russia     Moscow
IN   India  New Delhi
CH   China    Beijing
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

2.3 LÓGICA, CONTROL DE FLUJOS Y FILTROS

2.3.1 Operadores básicos

[866]:
```python
# Python no es capaz de comparar de objetos de diferentes tipos

my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])

print(my_house >= 18)
print(my_house < your_house)
```

```
[ True  True False False]
[False  True  True False]
```

2.3.2 Operadores booleanos

[867]:
```python
# and:

x = 12
print(x > 5 and x < 15)

# or:

```

```
y = 5
print(y < 7 or y > 13)

# not:

x1 = 8
y1 = 9
not(not(x < 3) and not(y > 14 or y > 10))

# Para usar los operadores en una matriz, es necesario utilizar los comandos␣
 ↪"logical_and()"m "logical_or()" y
# "logical_not()"

import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])

print(np.logical_or(my_house > 18.5, my_house < 10))

print(np.logical_and(my_house < 11, your_house < 11))
```

```
True
True
[False  True False  True]
[False False False  True]
```

2.3.3 If, elif, else

[868]:
```
z = 4

if z % 2 == 0:
    print("z is even")

###

z1 = 7

if z1 % 2 == 0:
    print("z is even")
else:
    print("z is odd")

###

z2 = 9

if z2 % 2 == 0:
```

```
    print("z is divisible by 2")
elif z2 % 3 == 0:
    print("z is divisible by 3")
else:
    print("z is neither divisible by 2 nor by 3")

# Nótese el caso con 6, el cual es divisible entre ambos: Python arrojará la␣
 ↪PRIMERA condición que se cumpla
```

```
z is even
z is odd
z is divisible by 3
```

2.3.4 Filtrando dataframes de pandas

[869]:
```
print(brics)

is_huge = brics.loc[:, "area"] > 8

print(brics[is_huge])

# Pero también se puede hacer en una sola línea de código:

print(brics[brics.loc[:, "area"] > 8])

# Para dos operadores booleanos:

print(brics[np.logical_and(brics.loc[:, "area"] > 8, brics.loc[:, "area"] <␣
 ↪10)])
```

```
        country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
   country   capital    area  population
BR  Brazil  Brasilia   8.516       200.4
RU  Russia    Moscow  17.100       143.5
CH   China   Beijing   9.597      1357.0
   country   capital    area  population
BR  Brazil  Brasilia   8.516       200.4
RU  Russia    Moscow  17.100       143.5
CH   China   Beijing   9.597      1357.0
   country   capital   area  population
BR  Brazil  Brasilia  8.516       200.4
CH   China   Beijing  9.597      1357.0
```

2.4 BUCLES

2.4.1 while

Es relativamente similar al if, elif, else: ejecuta el código mientras una condición verdadera, pero no se detendrá. Su sintaxis es:

while condition: expression

Es útil para repetir una acción hasta que se cumpla una condición

[870]:
```
error = 50.0

while error > 1:
    error = error / 4

print(error)
```

```
0.78125
```

[871]:
```
offset = 8

while offset != 0:
    print("correcting...")
    offset = offset - 1
    print(offset)
```

```
correcting…
7
correcting…
6
correcting…
5
correcting…
4
correcting…
3
correcting…
2
correcting…
1
correcting…
0
```

[872]:
```
offset = -7

while offset != 0 :
    print("correcting...")
    if offset > 0:
      offset = offset - 1
```

```
    else:
      offset = offset +1
    print(offset)
```

```
correcting…
-6
correcting…
-5
correcting…
-4
correcting…
-3
correcting…
-2
correcting…
-1
correcting…
0
```

2.4.2 for

Su sintaxis es:

for var in seq: expression

[873]:
```
fam = [1.73, 1.68, 1.71, 1.89]

for height in fam: # "height" es un nombre arbitrario
    print(height)
```

```
1.73
1.68
1.71
1.89
```

[874]:
```
for index, height in enumerate(fam):
    print("index " + str(index) + ": " + str(height))
```

```
index 0: 1.73
index 1: 1.68
index 2: 1.71
index 3: 1.89
```

[875]:
```
for c in "family":
    print(c.capitalize())
```

```
F
A
M
I
```

```
L
Y
```

[876]:
```
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

for index, area in enumerate(areas) :
    print("room " + str(index + 1) + ": " + str(area))
```

```
room 1: 11.25
room 2: 18.0
room 3: 20.0
room 4: 10.75
room 5: 9.5
```

[877]:
```
house = [["hallway", 11.25],
         ["kitchen", 18.0],
         ["living room", 20.0],
         ["bedroom", 10.75],
         ["bathroom", 9.50]]

for x in house:
    print("the " + str(x[0]) + " is " + str(x[1]) + " sqm")
```

```
the hallway is 11.25 sqm
the kitchen is 18.0 sqm
the living room is 20.0 sqm
the bedroom is 10.75 sqm
the bathroom is 9.5 sqm
```

2.4.3 Estructuras de datos en bucle 1

Para iterar sobre pares valor-clave en un diccionario, es necesario usar items() en el diccionario para definir la secuencia en el bucle for:

[878]:
```
world = {
    "afghanistan": 30.55,
    "albania": 2.77,
    "algeria": 39.21
}

for key, value in world.items():
    print(key + " -- " + str(value))
```

```
afghanistan -- 30.55
albania -- 2.77
algeria -- 39.21
```

[879]:
```
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
bmi = np_weight / np_height ** 2

for val in bmi:
    print(val)
```

```
21.85171572722109
20.97505668934241
21.750282138093777
24.74734749867025
21.44127836209856
```

Si se desea iterar sobre todos los ementos de una matriz NumPy, debe usarse nditer() para especificar la secuencia:

[880]:
```
meas = np.array([np_height, np_weight])

for val in np.nditer(meas):
    print(val)
```

```
1.73
1.68
1.71
1.89
1.79
65.4
59.2
63.6
88.4
68.7
```

[881]:
```
# Ejemplo capitales

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':
 ↪'vienna' }

for key, value in europe.items():
    print("the capital of " + str(key) + " is " + str(value))
```

```
the capital of spain is madrid
the capital of france is paris
the capital of germany is berlin
the capital of norway is oslo
the capital of italy is rome
the capital of poland is warsaw
the capital of austria is vienna
```

2.4.4 Estructuras de datos en bucle 2

En un dataframe Pandas, es necesario especificar si se desea iterar sobre columnas o renglones

[882]:
```
for lab, row in brics.iterrows():
    print(lab)
    print(row)
```

```
BR
country         Brazil
capital       Brasilia
area             8.516
population       200.4
Name: BR, dtype: object
RU
country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object
IN
country           India
capital       New Delhi
area              3.286
population       1252.0
Name: IN, dtype: object
CH
country         China
capital       Beijing
area            9.597
population     1357.0
Name: CH, dtype: object
SA
country       South Africa
capital           Pretoria
area                 1.221
population           52.98
Name: SA, dtype: object
```

[883]:
```
for lab, row in brics.iterrows():
    print(lab + ": " + row["capital"])
```

```
BR: Brasilia
RU: Moscow
IN: New Delhi
CH: Beijing
SA: Pretoria
```

[884]:
```
for lab, row in brics.iterrows():
    brics.loc[lab, "name_lenght"] = len(row["country"])
print(brics)
```

```
         country    capital    area  population  name_lenght
```

```
BR        Brazil   Brasilia   8.516      200.40          6.0
RU        Russia     Moscow  17.100      143.50          6.0
IN         India  New Delhi   3.286     1252.00          5.0
CH         China    Beijing   9.597     1357.00          5.0
SA  South Africa   Pretoria   1.221       52.98         12.0
```

Un mejor enfoque de lo anterior (que ahorra tiempo en bases de datos muy grandes) es con apply:

[885]:
```
brics["name_length1"] = brics["country"].apply(len)
print(brics)
```

```
         country    capital    area  population  name_lenght  name_length1
BR        Brazil   Brasilia   8.516      200.40          6.0             6
RU        Russia     Moscow  17.100      143.50          6.0             6
IN         India  New Delhi   3.286     1252.00          5.0             5
CH         China    Beijing   9.597     1357.00          5.0             5
SA  South Africa   Pretoria   1.221       52.98         12.0            12
```

2.5 ESTADÍSTICAS HACKER

2.5.1 Números aleatorios

[886]:
```
np.random.seed(123)

coin = np.random.randint(0, 2)
print(coin)

if coin == 0:
    print("heads")
else:
    print("tails")

print(np.random.randint(1, 7))
print(np.random.randint(1, 7))
```

```
0
heads
6
3
```

[887]:
```
step = 50

dice = np.random.randint(1,7)

if dice <= 2 :
    step = step - 1
elif dice <= 5 :
    step = step + 1
else :
```

```
    step = step + np.random.randint(1,7)

print(dice)
print(step)

# Cambia con cada corrida
```

```
5
51
```

2.5.2 Caminata aleatoria

[888]:
```
outcomes = []

for x in range(10):
    coin = np.random.randint(0, 2)
    if coin == 0:
        outcomes.append("heads")
    else:
        outcomes.append("tails")

print(outcomes)

# Aunque esto no es una caminata aleatoria, dado que los elementos de la lista␣
↪no se basan en los anteriores
```

```
['heads', 'heads', 'tails', 'tails', 'heads', 'tails', 'tails', 'heads',
'tails', 'heads']
```

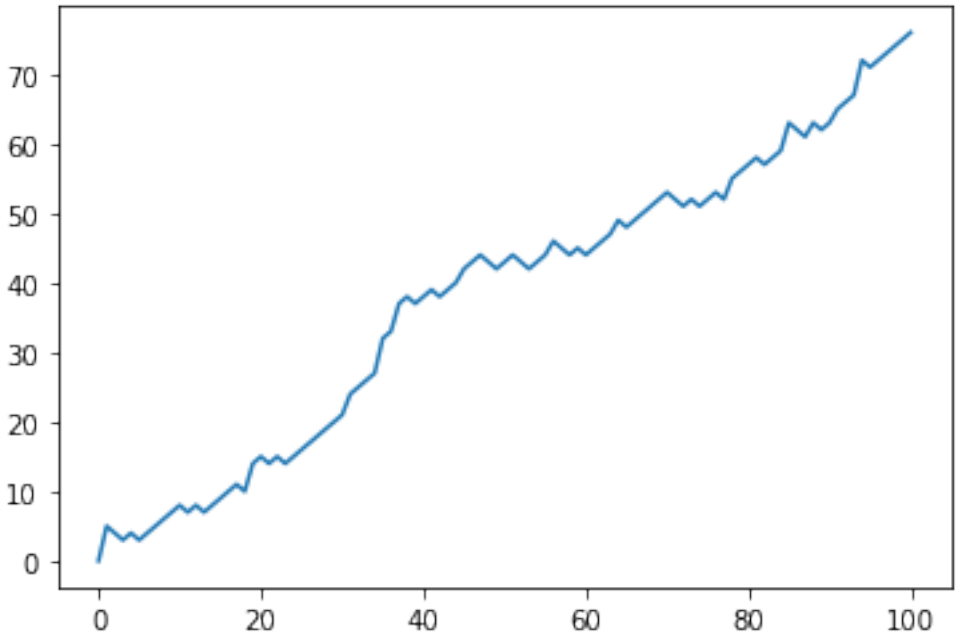
[889]:
```
# Rastreando el número total de tails mientras se simula el juego, se convierte␣
↪en caminata aleatoria:

tails = [0]

for x in range(10):
    coin = np.random.randint(0, 2)
    tails.append(tails[x] + coin)

print(tails)
```

```
[0, 1, 1, 2, 3, 3, 3, 3, 4, 5, 6]
```

[890]:
```
# Initialize random_walk
random_walk = [0]

for x in range(100) :
    # Set step: last element in random_walk
    step = random_walk[-1]
```

```
    # Roll the dice
    dice = np.random.randint(1,7)

    # Determine next step
    if dice <= 2:
        step = max(0, step - 1)
    elif dice <= 5:
        step = step + 1
    else:
        step = step + np.random.randint(1,7)

    # append next_step to random_walk
    random_walk.append(step)

# Print random_walk
print(random_walk)

# Gráfica

plt.plot(random_walk)

# Show the plot
plt.show()
```

```
[0, 5, 4, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 7, 8, 9, 10, 11, 10, 14, 15, 14, 15, 14,
15, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 32, 33, 37, 38, 37, 38, 39, 38, 39,
40, 42, 43, 44, 43, 42, 43, 44, 43, 42, 43, 44, 46, 45, 44, 45, 44, 45, 46, 47,
49, 48, 49, 50, 51, 52, 53, 52, 51, 52, 51, 52, 53, 52, 55, 56, 57, 58, 57, 58,
59, 63, 62, 61, 63, 62, 63, 65, 66, 67, 72, 71, 72, 73, 74, 75, 76]
```



2.5.3 Distribución

Finalmente, calcularemos muchas caminatas aleatorias para conocer su distribución

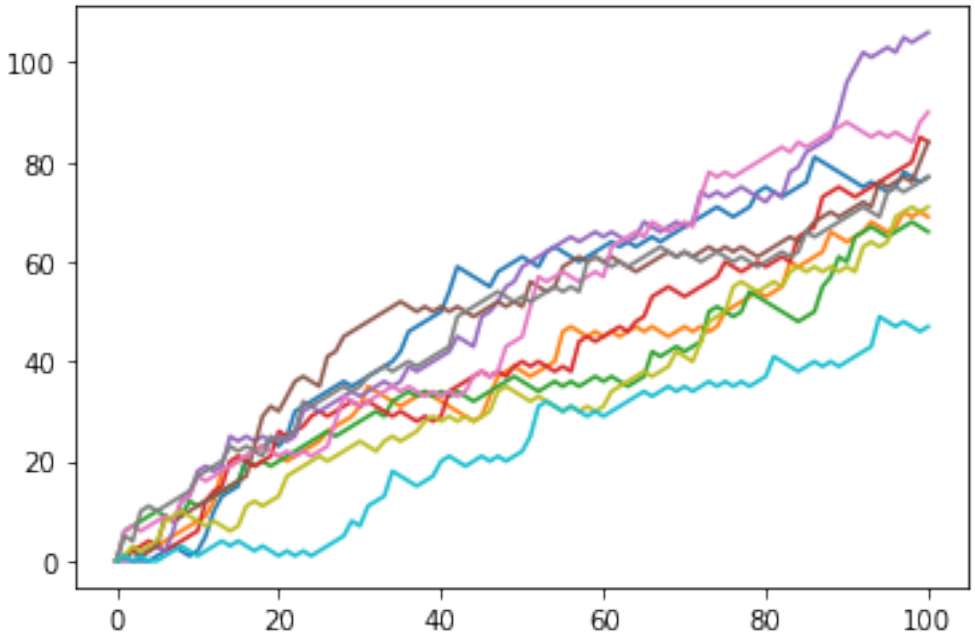
[891]:
```
# Initialize all_walks
all_walks = []

# Simulate random walk 10 times
for i in range(10) :

    # Code from before
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)

        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)
        random_walk.append(step)

    # Append random_walk to all_walks
    all_walks.append(random_walk)
```

```
# Print all_walks
print(all_walks)
```

```
[[0, 0, 5, 6, 7, 8, 7, 6, 5, 4, 3, 4, 5, 6, 12, 11, 10, 9, 10, 11, 12, 11, 12,
13, 12, 17, 18, 17, 16, 15, 14, 15, 14, 13, 15, 16, 19, 20, 22, 23, 24, 25, 24,
23, 24, 25, 26, 27, 26, 27, 28, 29, 32, 33, 34, 33, 34, 33, 34, 35, 36, 40, 39,
38, 39, 40, 41, 40, 41, 40, 41, 43, 48, 47, 48, 49, 50, 51, 52, 53, 52, 53, 54,
55, 57, 59, 58, 59, 60, 63, 62, 63, 62, 63, 64, 65, 64, 65, 64, 65, 66], [0, 2,
1, 0, 1, 2, 1, 3, 4, 5, 6, 7, 8, 9, 8, 9, 10, 15, 16, 15, 14, 13, 12, 13, 14,
15, 20, 21, 20, 21, 22, 23, 24, 25, 26, 25, 24, 25, 24, 30, 29, 30, 29, 30, 29,
30, 34, 38, 40, 42, 41, 42, 44, 43, 49, 50, 52, 53, 52, 53, 54, 55, 54, 55, 56,
57, 59, 58, 57, 58, 64, 69, 68, 67, 71, 72, 73, 74, 76, 75, 76, 75, 76, 75, 74,
80, 79, 80, 81, 82, 83, 84, 85, 86, 87, 86, 87, 86, 85, 86, 85], [0, 1, 0, 1, 2,
6, 7, 6, 12, 11, 10, 11, 10, 9, 10, 9, 13, 12, 13, 12, 13, 12, 13, 15, 14, 13,
12, 13, 12, 17, 18, 19, 18, 17, 18, 17, 18, 17, 22, 23, 22, 21, 20, 19, 20, 21,
25, 31, 32, 33, 34, 35, 34, 35, 37, 38, 39, 40, 39, 38, 39, 38, 39, 40, 41, 40,
41, 42, 43, 42, 41, 42, 43, 48, 49, 50, 55, 54, 55, 54, 55, 56, 57, 58, 57, 63,
62, 63, 62, 61, 62, 66, 67, 68, 69, 70, 69, 68, 67, 66, 65], [0, 1, 2, 3, 4, 5,
6, 5, 6, 7, 6, 7, 8, 11, 10, 11, 10, 11, 14, 15, 16, 17, 16, 15, 18, 19, 20, 21,
20, 19, 20, 23, 24, 29, 30, 29, 30, 29, 30, 29, 35, 34, 33, 34, 36, 35, 34, 33,
34, 35, 36, 37, 41, 46, 47, 48, 49, 52, 53, 54, 55, 56, 57, 58, 57, 56, 57, 58,
57, 63, 62, 63, 64, 63, 62, 63, 62, 61, 67, 66, 65, 66, 67, 66, 65, 64, 63, 64,
70, 71, 74, 75, 81, 82, 83, 89, 88, 94, 93, 94, 95], [0, 1, 0, 6, 7, 6, 7, 6, 7,
6, 7, 6, 7, 11, 12, 14, 13, 14, 13, 12, 13, 12, 13, 12, 16, 20, 19, 18, 17, 18,
19, 23, 24, 25, 29, 28, 27, 26, 27, 28, 30, 29, 28, 29, 30, 32, 35, 40, 41, 40,
41, 42, 43, 44, 45, 44, 45, 46, 52, 51, 56, 55, 59, 58, 62, 61, 64, 65, 64, 65,
71, 70, 69, 68, 72, 71, 70, 71, 72, 71, 72, 73, 74, 75, 74, 73, 72, 73, 72, 71,
75, 76, 78, 82, 88, 87, 86, 87, 88, 89, 88], [0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 16, 15, 18, 19, 18, 17, 18, 19, 20, 19, 18, 17, 16, 17, 18,
19, 20, 21, 22, 23, 22, 23, 24, 26, 25, 26, 27, 30, 32, 33, 32, 31, 30, 29, 28,
29, 28, 29, 31, 32, 33, 32, 33, 32, 31, 32, 31, 30, 29, 30, 29, 28, 29, 30, 31,
33, 34, 35, 34, 33, 34, 40, 39, 40, 41, 46, 45, 46, 45, 46, 47, 46, 47, 48, 54,
55, 56, 57, 56, 58, 59, 60, 61, 60], [0, 1, 2, 3, 4, 3, 4, 3, 4, 5, 4, 5, 9, 10,
13, 14, 18, 19, 18, 19, 20, 19, 18, 23, 24, 25, 24, 23, 24, 25, 26, 25, 29, 28,
29, 30, 31, 32, 33, 34, 35, 34, 33, 32, 33, 34, 38, 39, 38, 39, 40, 39, 45, 46,
47, 48, 47, 48, 49, 52, 51, 52, 51, 50, 49, 50, 51, 53, 54, 55, 59, 60, 59, 58,
59, 60, 59, 58, 59, 60, 61, 60, 59, 60, 61, 62, 61, 62, 63, 62, 61, 60, 61, 62,
63, 64, 65, 70, 74, 73, 74], [0, 0, 0, 0, 1, 0, 1, 2, 1, 5, 6, 5, 6, 5, 4, 5, 4,
8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16, 21, 22, 23, 22, 23, 22, 23,
24, 23, 24, 29, 30, 31, 30, 31, 32, 33, 32, 33, 32, 33, 34, 33, 32, 31, 32, 33,
32, 31, 30, 31, 35, 36, 37, 38, 39, 40, 39, 40, 41, 40, 41, 46, 47, 46, 45, 46,
45, 44, 43, 47, 48, 49, 50, 49, 50, 51, 52, 53, 54, 53, 52, 55, 54, 53, 52, 51,
52, 58, 59, 58], [0, 1, 2, 1, 0, 1, 2, 3, 2, 6, 7, 6, 8, 9, 8, 9, 10, 11, 12,
13, 14, 13, 12, 13, 14, 13, 14, 15, 14, 13, 12, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 24, 25, 26, 27, 28, 29, 28, 27, 33, 34, 35, 34, 33, 32, 36, 37, 41, 40,
46, 47, 48, 52, 53, 52, 51, 52, 53, 54, 55, 54, 60, 61, 60, 64, 65, 64, 65, 64,
70, 76, 77, 76, 77, 78, 79, 80, 81, 80, 83, 82, 84, 85, 91, 90, 91, 92, 91, 90,
```

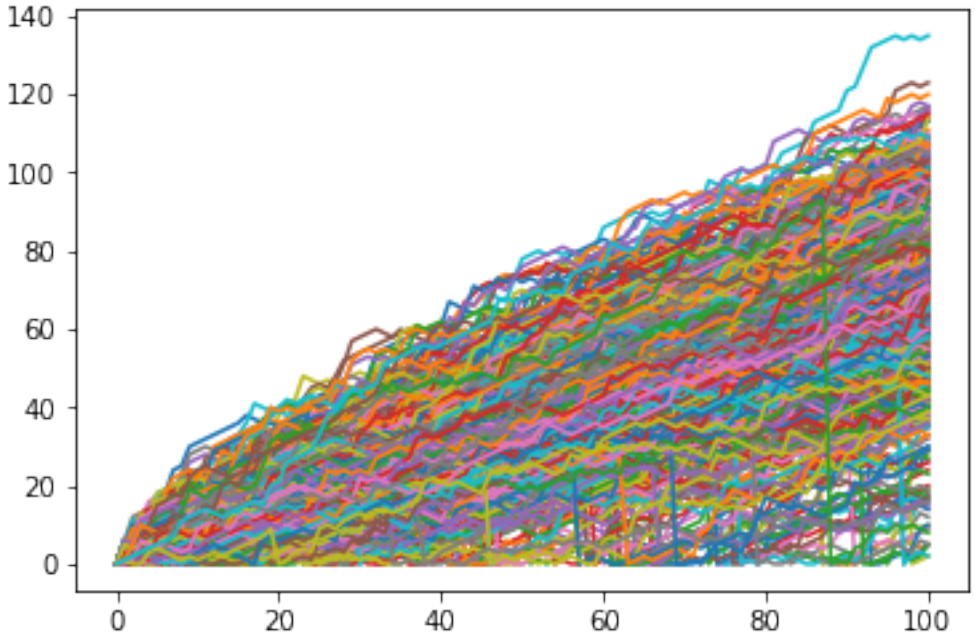
```
91, 90], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 13, 14, 15, 16, 15,
14, 13, 14, 13, 17, 16, 15, 16, 18, 19, 24, 23, 24, 25, 29, 28, 29, 30, 29, 28,
29, 30, 31, 30, 29, 28, 29, 28, 27, 28, 27, 26, 25, 26, 27, 26, 25, 24, 25, 26,
27, 26, 25, 26, 27, 28, 29, 28, 29, 28, 29, 30, 31, 32, 33, 38, 43, 42, 43, 44,
43, 44, 50, 49, 50, 49, 50, 49, 50, 51, 52, 51, 50, 51, 52, 53, 52, 51, 50, 52,
53]]
```

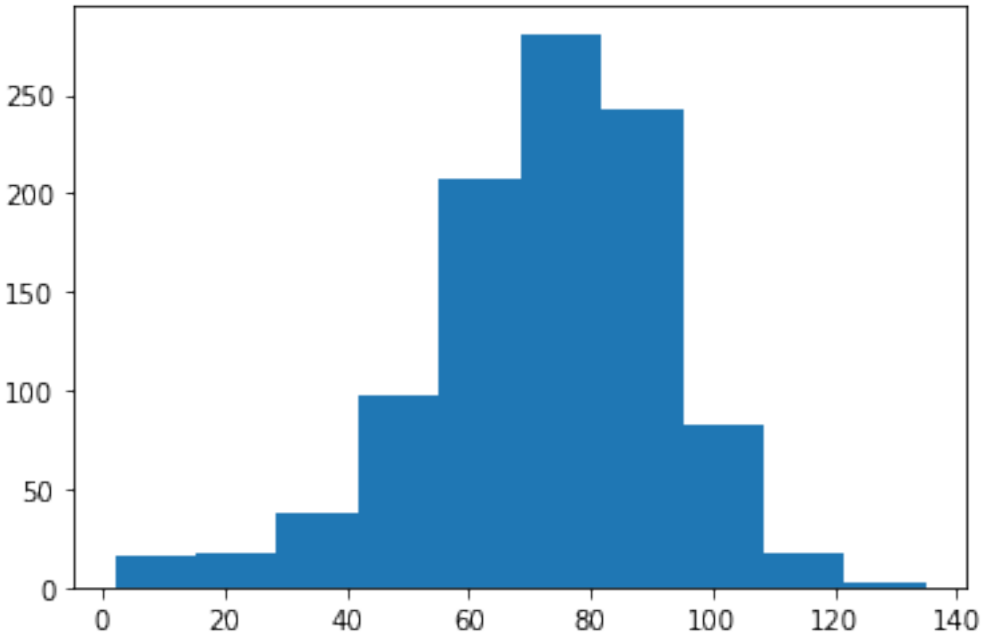
[892]:
```python
# initialize and populate all_walks
all_walks = []
for i in range(10) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)
        random_walk.append(step)
    all_walks.append(random_walk)

# Convert all_walks to NumPy array: np_aw
np_aw = np.array(all_walks)

# Transpose np_aw: np_aw_t
np_aw_t = np.transpose(np_aw)

# Plot np_aw_t and show
plt.plot(np_aw_t)
plt.show()
```



[893]:
```
# Simulate random walk 250 times
all_walks = []
for i in range(1000) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)

        # Implement clumsiness
        if np.random.rand() <= 0.001 :
            step = 0

        random_walk.append(step)
    all_walks.append(random_walk)

# Create and plot np_aw_t
np_aw_t = np.transpose(np.array(all_walks))
plt.plot(np_aw_t)
plt.show()
```

```
###

ends = np_aw_t[-1, :]

# Plot histogram of ends, display plot
plt.hist(ends)
plt.show()
```





3 PYTHON DATA SCIENCE TOOLBOX 1

3.1 ESCRIBIENDO FUNCIONES PROPIAS

Python tiene funciones predeterminadas como: str():

[894]:
```
x = str(5)
print(x)
print(type(x))
```

```
5
<class 'str'>
```

Supóngase que se quiere definir una función para elevar un número al cuadrado:

[895]:
```
def square(value):
    new_value = value ** 2
    print(new_value)

square(9)

# Si no se quiere imprimir el resultado, y en su lugar se requiere guardarlo en␣
 ↪algún objeto:

def square1(value1):
    new_value1 = value1 **2
    return new_value1
```

```
num = square(6)
```

```
81
36
```

Docstrings describen qué hace la función, y sirven como la documentación de esta.

[896]:
```
def square(value):
    """Return de square of a value-"""
    new_value = value ** 2
    return new_value

square(4)
```

[896]: 16

[897]:
```
def shout(word):
    """Print a string with three exclamation marks"""
    shout_word = word + '!!!'
    print(shout_word)

shout("congratulations")
```

```
congratulations!!!
```

[898]:
```
# Alternativamente:

def shout(word):
    """Return a string with three exclamation marks"""
    shout_word = word + "!!!"
    return shout_word

yell = shout("congratulations")

print(yell)
```

```
congratulations!!!
```

3.1.1 Funciones con múltiples parámetros

[899]:
```
def raise_to_power(value1, value2):
    """Raise value one to the power of value2."""
    new_value = value1 ** value2
    return new_value

result = raise_to_power(4, 3)
print(result)
```

```
64
```

[900]:
```
### Tuplas

even_nums = (2, 4, 6)
a, b, c = even_nums
print(a)
print(b)
print(c)
print(even_nums[1])
```

```
2
4
6
4
```

[901]:
```
### Ahora, modificaremos la función para que arroje el valor 1 elevado al valor␣
↪2 y viceversa:

def raise_both(value1, value2):
    """Raise value1 to the power of value2 and viceversa."""
    new_value1 = value1 ** value2
    new_value2 = value2 ** value1
    new_tuple = (new_value1, new_value2)
    return new_tuple

result = raise_both(2, 3)
print(result)
```

```
(8, 9)
```

[902]:
```
def shout(word1, word2):
    """Concatenate strings with three exclamation marks"""
    shout1 = word1 + "!!!"
    shout2 = word2 + "!!!"
    new_shout = shout1 + shout2
    return new_shout

yell = shout("congratulations", "you")
print(yell)
```

```
congratulations!!!you!!!
```

[903]:
```
def shout_all(word1, word2):
    shout1 = word1 + "!!!"
    shout2 = word2 + "!!!"
    shout_words = (shout1, shout2)
    return shout_words

yell1, yell2 = shout_all("congratulations", "you")
print(yell1)
```

```
print(yell2)
```

```
congratulations!!!
you!!!
```

[904]:
```
### Ejemplo Twitter

twitter = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/tweets.csv")

langs_count = {}

col = twitter['lang']

for entry in col:
    if entry in langs_count.keys():
        langs_count[entry] = langs_count[entry] + 1
    else:
        langs_count[entry] = 1

print(langs_count)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

[905]:
```
def count_entries(twitter, col_name):
    """Return a dictionary with counts of
    occurrences as value for each key."""

    langs_count = {}

    col = twitter[col_name]

    for entry in col:
        if entry in langs_count.keys():
            langs_count[entry] = langs_count[entry] + 1
        else:
            langs_count[entry] = 1
    return langs_count

result = count_entries(twitter, "lang")
print(result)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

3.2 ARGUMENTOS DEFAULT, LONGITUD DE VARIABLE Y ALCANCE

El alcance es la parte de un programa donde un objeto o nombre puede ser accesible. Hay tres tipos:

- Global: definido en el cuerpo principal de un script;
- Local: definido dentro de una función;

- Built-in: nombres en modulos predefinidos de Python

[906]:
```
def square(value):
    """Returns the square of a number."""
    new_val = value ** 2
    return new_val

print(square(3))

# print(new_val) # no es accesible porque se definió solo dentro del ámbito␣
 ↪local de la función
```

```
9
```

[907]:
```
# En cambio, si se define el nombre globalmente antes de definir y llamar a la␣
 ↪función:

new_val1 = 10

def square(value1):
    """Returns the square of a number."""
    new_val1 = value1 ** 2
    return new_val1

print(new_val1)

# Al buscar algo, Python primero lo hace en el ámbito local, luego en el global␣
 ↪y, si no se encuentra, en algún built-in
```

```
10
```

[908]:
```
# Para cambiar un nombre global dentro de una función, usamos global:

new_val2 = 10

def square(value2):
    """TReturns the square of a number."""
    global new_val2
    new_val2 = new_val2 ** 2
    return new_val2

print(new_val2)
```

```
10
```

[909]:
```
team = "teen titans"

def change_team():
    """Change the value of the global variable team."""
```

```
    global team
    team = "justice league"
    print(team)

change_team()
print(team)
```

```
justice league
justice league
```

3.2.1 Funciones anidadas

Su sintaxis es como sigue:

```
def outer(...):
""" ... """
x = ...

def inner( ... ):
""" ... """
y = x ** 2

return ...
```

[910]:
```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""

    def inner(x):
        """Returns the remainder plus 5 of a value"""
        return x % 2 + 5

    return(inner(x1), inner(x2), inner(x3))

print(mod2plus5(1, 2, 3))
```

```
(6, 5, 6)
```

[911]:
```
def raise_val(n):
    """Return the inner function."""

    def inner(x):
        """Raise x to the power of n."""
        raised = x ** n
        return raised

    return inner

square = raise_val(2)
cube = raise_val(3)
```

```
print(square(2), cube(4))

# En una función anidada, se puede usar "nonlocal" para crear y cambiar nombres␣
 ↪en un ámbito adjunto
```

```
4 64
```

[912]:
```
def three_shouts(word1, word2, word3):
    """Returns a tuple of strings
    concatenated with '!!!!'."""

    def inner(word):
        """Returns a string concatenated with '!!!!'."""
        return word + '!!!!'

    return (inner(word1), inner(word2), inner(word3))

print(three_shouts('a', 'b', 'c'))
```

```
('a!!!!', 'b!!!!', 'c!!!!')
```

[913]:
```
def echo(n):
    """Return the inner_echo function."""

    def inner_echo(word1):
        """Concatenate n copies of word1."""
        echo_word = word1 * n
        return echo_word

    return inner_echo

twice = echo(2)
thrice = echo(3)
print(twice('hello'), thrice('hello'))
```

```
hellohello hellohellohello
```

[914]:
```
def echo_shout(word):
    """Change the value of a nonlocal variable"""

    echo_word = word + word

    print(echo_word)

    def shout():
        """Alter a variable in the enclosing scope"""
        nonlocal echo_word

        echo_word = echo_word + "!!!"
```

```
    shout()

    print(echo_word)

echo_shout("hello")
```

```
hellohello
hellohello!!!
```

3.2.2 Argumentos flexibles y default

[915]:
```
# Una función con un argumento por default es:

def power(number, pow = 1):
    """Raise number to the power of pow."""
    new_value = number ** pow
    return new_value

power(9, 2) # si especificamos el 2do argumento, la función lo sobreescribe en␣
 ↪el default, pero si no:

power(9)
```

[915]:
```
9
```

[916]:
```
# Ejemplo default:

def shout_echo(word1, echo = 1):
    """Concatenate echo copies of word1 and three
     exclamation marks at the end of the string."""

    echo_word = word1 * echo

    shout_word = echo_word + '!!!'

    return shout_word

no_echo = shout_echo("Hey")

with_echo = shout_echo("Hey", echo = 5)

print(no_echo)
print(with_echo)
```

```
Hey!!!
HeyHeyHeyHeyHey!!!
```

[917]:
```
def shout_echo(word1,echo = 1, intense = False):
    """Concatenate echo copies of word1 and three
    exclamation marks at the end of the string."""

    echo_word = word1 * echo

    if intense is True:
        echo_word_new = echo_word.upper() + '!!!!'
    else:
        echo_word_new = echo_word + '!!!!'

    return echo_word_new

with_big_echo = shout_echo("Hey", 5, True)

big_no_echo = shout_echo("Hey", intense = True)

print(with_big_echo)
print(big_no_echo)
```

```
HEYHEYHEYHEYHEY!!!!
HEY!!!!
```

[918]:
```
# Ejemplo flexibles:

def gibberish(*args):
    """Concatenate strings in *args together."""

    hodgepodge = ""

    for word in args:
        hodgepodge += word

    return hodgepodge

one_word = gibberish("luke")

many_words = gibberish("luke", "leia", "han", "obi", "darth")

print(one_word)
print(many_words)
```

```
luke
lukeleiahanobidarth
```

[919]:
```
# kwargs permite pasar un número de argumentos de clave a funciones:

def report_status(**kwargs):
```

```
    """Print out the status of a movie character."""

    print("\nBEGIN: REPORT\n")

    for key, value in kwargs.items():
        print(key + ": " + value)

    print("\nEND REPORT")

report_status(name = "luke", affiliation = "jedi", status = "missing")

report_status(name="anakin", affiliation="sith lord", status="deceased")
```

```
BEGIN: REPORT

name: luke
affiliation: jedi
status: missing

END REPORT

BEGIN: REPORT

name: anakin
affiliation: sith lord
status: deceased

END REPORT
```

[920]:
```
# Resumen

tweets_df = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/tweets.csv")

def count_entries(df, col_name='lang'):
    """Return a dictionary with counts of
    occurrences as value for each key."""
    cols_count = {}
    col = df[col_name]
    for entry in col:
        if entry in cols_count.keys():
            cols_count[entry] += 1
        else:
            cols_count[entry] = 1
    return cols_count
result1 = count_entries(tweets_df,col_name='lang')
result2 = count_entries(tweets_df,col_name='source')
```

```
print(result1)
print(result2)
```

```
{'en': 97, 'et': 1, 'und': 2}
{'<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>': 24, '<a
href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a
href="http://twitter.com/download/android" rel="nofollow">Twitter for
Android</a>': 26, '<a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com"
rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/"
rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad"
rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com"
rel="nofollow">Linkis.com</a>': 2, '<a
href="http://rutracker.org/forum/viewforum.php?f=93"
rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com"
rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/"
rel="nofollow">Plume\xa0for\xa0Android</a>': 1}
```

[921]:
```
def count_entries(df, *args):
    """Return a dictionary with counts of
    occurrences as value for each key."""

    cols_count = {}

    for col_name in args:

        col = df[col_name]

        for entry in col:

            if entry in cols_count.keys():
                cols_count[entry] += 1

            else:
                cols_count[entry] = 1

    return cols_count

result1 = count_entries(tweets_df, "lang")

result2 = count_entries(tweets_df, "lang", "source")

print(result1)
print(result2)
```

```
{'en': 97, 'et': 1, 'und': 2}
{'en': 97, 'et': 1, 'und': 2, '<a href="http://twitter.com"
rel="nofollow">Twitter Web Client</a>': 24, '<a
```

```
href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a
href="http://twitter.com/download/android" rel="nofollow">Twitter for
Android</a>': 26, '<a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com"
rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/"
rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad"
rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com"
rel="nofollow">Linkis.com</a>': 2, '<a
href="http://rutracker.org/forum/viewforum.php?f=93"
rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com"
rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/"
rel="nofollow">Plume\xa0for\xa0Android</a>': 1}
```

3.3 FUNCIONES LAMBDA Y MANEJO DE ERRORES

Las funciones lambda para escribir funciones de manera más compacta y rápida.

[922]:
```
raise_to_power = lambda x, y: x ** y

raise_to_power(2, 3)
```

[922]:
```
8
```

[923]:
```
nums = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, nums)

print(list(square_all))
```

```
[2304, 36, 81, 441, 1]
```

[924]:
```
echo_word = lambda word1, echo: word1 * echo

result = echo_word("hey", 5)

print(result)
```

```
heyheyheyheyhey
```

[925]:
```
spells = ["protego", "accio", "expecto patronum", "legilimens"]

shout_spells = map(lambda item: item + "!!!" , spells)

shout_spells_list = list(shout_spells)

print(shout_spells_list)
```

```
['protego!!!', 'accio!!!', 'expecto patronum!!!', 'legilimens!!!']
```

[926]:
```
fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir',␣
↪'legolas', 'gimli', 'gandalf']

result = filter(lambda member: len(member) > 6, fellowship)

result_list = list(result)

print(result_list)
```

```
['samwise', 'aragorn', 'boromir', 'legolas', 'gandalf']
```

La función reduce() es útil para calcular algún cómputo en una lista, y a diferencia de map() y filter(), arroja un solo valor como resultado. Nota: debe importarse del módulo functools.

[927]:
```
from functools import reduce

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

result = reduce(lambda item1, item2: item1 + item2, stark)

print(result)
```

```
robbsansaaryabrandonrickon
```

3.3.1 Introducción a manejo de errores

Considérese el siguiente ejemplo:

[928]:
```
def sqrt(x):
    """Returns the square root of a number."""
    return x ** (0.5)

print(sqrt(10))

# print(sqrt("hello")) arrojará un error
```

```
3.1622776601683795
```

Errores y excepciones:

- Excepciones: durante la ejecución;
  - Pueden tomarse excepciones con una cláusula try-except, en donde Python trata de correr el código después de "try"
  - Si existe una excepción, ejecuta el código después de de "except"

[929]:
```
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except:
```

```
        print("x must be an int or float")

print(sqrt(4))
print(sqrt("hi"))

# Puede ser que solo interesen los TypeErros, en cuyo caso debería escribirse␣
 ↪"except TypeError:"
```

```
2.0
x must be an int or float
None
```

[930]:
```
# También puede quererse  generar un error mediante la clave de aumentar

# Supóganse que no se quiere que la función anterior funcione para números␣
 ↪negativos:

def sqrt1(x):
    """Returns the square root of a number."""
    if x < 0:
        raise ValueError("x must be non-negative")
    try:
        return x ** 0.5
    except TypeError:
        print("x must be an int or float")

# print(sqrt1(-16)) arrojará un error
```

[931]:
```
# Ejemplos

def shout_echo(word1, echo=1):
    """Concatenate echo copies of word1 and three
    exclamation marks at the end of the string."""

    echo_words = ""
    shout_words = ""

    try:
        echo_word = word1 * echo

        shout_words = echo_word + "!!!"
    except:
        print("word1 must be a string and echo must be an integer.")

    return shout_words

shout_echo("particle", echo="accelerator")
```

```
shout_echo("hola", echo = 7)
```

word1 must be a string and echo must be an integer.

[931]: 'holaholaholaholaholaholahola!!!'

[932]:
```
def shout_echo(word1, echo=1):
    """Concatenate echo copies of word1 and three
    exclamation marks at the end of the string."""

    if echo < 0:
        raise ValueError("echo must be greater than or equal to 0")

    echo_word = word1 * echo

    shout_word = echo_word + '!!!'

    return shout_word

print(shout_echo("particle", echo=5))
# print(shout_echo("hello", echo = -8)) arrojará el error definido
```

particleparticleparticleparticleparticle!!!

[933]:
```
# Ejemplo

# Para identificar los retuits del dataframe:

result = filter(lambda x: x[0:2] == "RT", twitter["text"])

res_list = list(result)

for tweet in res_list:
    print(tweet)
```

```
RT @bpolitics: .@krollbondrating's Christopher Whalen says Clinton is the
weakest Dem candidate in 50 years https://t.co/pLk7rvoRSn https:/…
RT @HeidiAlpine: @dmartosko Cruz video found…racing from the scene…
#cruzsexscandal https://t.co/zuAPZfQDk3
RT @AlanLohner: The anti-American D.C. elites despise Trump for his America-
first foreign policy. Trump threatens their gravy train. https:…
RT @BIackPplTweets: Young Donald trump meets his neighbor
https://t.co/RFlu17Z1eE
RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective
amnisia.
RT @HouseCracka: 29,000+ PEOPLE WATCHING TRUMP LIVE ON ONE STREAM!!!

https://t.co/7QCFz9ehNe
RT @urfavandtrump: RT for Brendon Urie
```

Fav for Donald Trump https://t.co/PZ5vS94lOg
RT @trapgrampa: This is how I see #Trump every time he speaks.
https://t.co/fYSiHNSOnT
RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective
amnisia.
RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT
RAISING TAXES in WI! #LyinTrump
#NeverTrump #CruzCrew https…
RT @urfavandtrump: RT for Brendon Urie
Fav for Donald Trump https://t.co/PZ5vS94lOg
RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might
vote for Trump. A total fabrication, but whatever… https:/…
RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT
RAISING TAXES in WI! #LyinTrump
#NeverTrump #CruzCrew https…
RT @trapgrampa: This is how I see #Trump every time he speaks.
https://t.co/fYSiHNSOnT
RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump
at any time and Corey can do nothing? Michelle is clearly…
RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held
by Dems and GOP https://t.co/D7ulnAJhis   #tcot #PJNET https…
RT @DRUDGE_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who
has terminal illness… https://t.co/qt06aG9inT
RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might
vote for Trump. A total fabrication, but whatever… https:/…
RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't
know how great he is, advice watch live streaming https://…
RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held
by Dems and GOP https://t.co/D7ulnAJhis   #tcot #PJNET https…
RT @DRUDGE_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who
has terminal illness… https://t.co/qt06aG9inT
RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't
know how great he is, advice watch live streaming https://…
RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump
at any time and Corey can do nothing? Michelle is clearly…
RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either
to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…
RT @Norsu2: Nightmare WI poll for Ted Cruz has Kasich surging: Trump 29, Kasich
27, Cruz 25. https://t.co/lJsgbLYY1P #NeverTrump
RT @thehill: WATCH: Protester pepper-sprayed point blank at Trump rally
https://t.co/B5f65Al9ld https://t.co/skAfByXuQc
RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either
to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…
RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might
vote for Trump. A total fabrication, but whatever… https:/…
RT @DebbieStout5: Wow! Last I checked it was just 12 points & that wasn't
more than a day ago. Oh boy Trump ppl might want to rethink http…
RT @tyleroakley: i'm a messy bitch, but at least i'm not voting for trump

RT @vandives: Trump supporters r tired of justice NOT being served. There's no justice anymore. Hardworking Americans get screwed. That's n…
RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.
RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.
RT @urfavandtrump: RT for Jerrie (Little Mix)
Fav for Donald Trump https://t.co/nEVxElW6iG
RT @urfavandtrump: RT for Jerrie (Little Mix)
Fav for Donald Trump https://t.co/nEVxElW6iG
RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…
RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…
RT @AIIAmericanGirI: VIDEO=> Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face
https://t.co/GeEryMDuDY
RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…
RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump
URGENT PUBLIC TRUMP ALERT:
COVERT KILL MEANS https:…
RT @AIIAmericanGirI: VIDEO=> Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face
https://t.co/GeEryMDuDY
RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…
RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump
URGENT PUBLIC TRUMP ALERT:
COVERT KILL MEANS https:…
RT @Schneider_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…
RT @RonBasler1: @DavidWhitDennis @realDonaldTrump @tedcruz

CRUZ SCREWS HOOKERS

CRUZ / CLINTON
RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his college educat…
RT @Schneider_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…
RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because

```
of Trump pageant, her 7 yr. old son has his college educat…
RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes"
or "no" answer from Cruz a/b being unfaithful to his wife @T…
RT @RonBasler1: @DavidWhitDennis @realDonaldTrump @tedcruz

CRUZ SCREWS HOOKERS

CRUZ / CLINTON
RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world
hostage was an idea for a funny movie? #Trump #GOP https:/…
RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald
Trump… but [he] didn't come out of nowhere." -Hillary
https…
RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes"
or "no" answer from Cruz a/b being unfaithful to his wife @T…
RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald
Trump… but [he] didn't come out of nowhere." -Hillary
https…
RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world
hostage was an idea for a funny movie? #Trump #GOP https:/…
RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day,
from 8am to 10pm today, then the reruns come on, repeating t…
RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day,
from 8am to 10pm today, then the reruns come on, repeating t…
RT @GlendaJazzey: Donald Trump's Campaign Financing Dodge, @rrotunda
https://t.co/L8flI4lswG via @VerdictJustia
RT @TUSK81: LOUDER FOR THE PEOPLE IN THE BACK https://t.co/hlPVyNLXzx
RT @loopzoop: Well…put it back https://t.co/8Yb7BDT5VM
RT @claytoncubitt: Stop asking Bernie supporters if they'll vote for Hillary
against Trump. We got a plan to beat Trump already. Called Ber…
RT @akaMaude13: Seriously can't make this up. What a joke. #NeverTrump
https://t.co/JkTx6mdRgC
```

[934]:
```python
def count_entries(df, col_name='lang'):
    """Return a dictionary with counts of
    occurrences as value for each key."""

    cols_count = {}

    try:
        col = df[col_name]

        for entry in col:

            if entry in cols_count.keys():
                cols_count[entry] += 1
            else:
```

```
                cols_count[entry] = 1

        return cols_count

    except:
        print("The DataFrame does not have a ' + col_name + ' column.")

result1 = count_entries(twitter, 'lang')

print(result1)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

[935]:
```
def count_entries(df, col_name='lang'):
    """Return a dictionary with counts of
    occurrences as value for each key."""

    if col_name not in df.columns:
        raise ValueError("The DataFrame does not have a lang1 column.")

    cols_count = {}

    col = df[col_name]

    for entry in col:

        if entry in cols_count.keys():
            cols_count[entry] += 1
        else:
            cols_count[entry] = 1

    return cols_count

result1 = count_entries(twitter, "lang")

print(result1)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

4 PYTHON DATA SCIENCE TOOLBOX 2

4.1 Iteratores

Como se revisó, puede iterarse con un bucle for, o bien, sobre un objeto de rango.

Este tipo de objetos son iterables: listas, cadenas, diccionarios y conexiones de archivo. Estos tienen un método iter() asociado.

Por su lado, un iterador tiene un método asociado next() que produce el siguiente valor.

[936]:
```
word = "Hello"

it = iter(word)

print(next(it))
print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

```
H
e
l
l
o
```

[937]:
```
# Alternativamente:
word = "data"
it = iter(word)
print(*it)
```

```
d a t a
```

[938]:
```
# Para diccionarios:

pythonistas = {
    "hugo": "bowne-anderson",
    "francis": "castro"
}

for key, value in pythonistas.items():
    print(key,value)
```

```
hugo bowne-anderson
francis castro
```

[939]:
```
# Para archivos:

it = iter(twitter)
print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

```
contributors
coordinates
created_at
entities
```

[940]:
```
flash = ['jay garrick', 'barry allen', 'wally west', 'bart allen']


for person in flash:
    print(person)


superhero = iter(flash)

print(next(superhero))
print(next(superhero))
print(next(superhero))
print(next(superhero))
```

```
jay garrick
barry allen
wally west
bart allen
jay garrick
barry allen
wally west
bart allen
```

[941]:
```
small_value = iter(range(3))

print(next(small_value))
print(next(small_value))
print(next(small_value))

for num in range(3):
    print(num)

###
googol = iter(range(10**100))

print(next(googol))
print(next(googol))
print(next(googol))
print(next(googol))
print(next(googol))
print(next(googol))
print(next(googol))
print(next(googol))
```

```
0
1
2
0
```

```
1
2
0
1
2
3
4
5
6
7
```

[942]:
```
values = range(10, 21)

print(values)

values_list = list(values)

print(values_list)

values_sum = sum(values)

print(values_sum)
```

```
range(10, 21)
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
165
```

La fución enumerate permite agregar un contador a cualquier iterable.

[943]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]

e = enumerate(avengers)

e_list = list(e)

print(e_list)

# Alternativamente:

for index, value in enumerate(avengers, start = 1): # start indica en qué␣
↪número empieza la numeración
    print(index, value)
```

```
[(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')]
1 hawkeye
2 iron man
3 thor
4 quicksilver
```

La función zip permite unir un número arbitrario de iterables.

[944]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]
names = ["barton", "stark", "odinson", "maximoff"]

z = zip(avengers, names)

z_list = list(z)

print(z_list)

# Alternativamente:

for z1, z2 in zip(avengers, names):
    print(z1, z2)
```

```
[('hawkeye', 'barton'), ('iron man', 'stark'), ('thor', 'odinson'),
('quicksilver', 'maximoff')]
hawkeye barton
iron man stark
thor odinson
quicksilver maximoff
```

[945]:
```
# Ejemplo

mutants = ['charles xavier',
           'bobby drake',
           'kurt wagner',
           'max eisenhardt',
           'kitty pryde']

mutant_list = list(enumerate(mutants))

print(mutant_list)

for index1, value1 in enumerate(mutants):
    print(index1, value1)

for index2, value2 in enumerate(mutants, start = 1):
    print(index2, value2)
```

```
[(0, 'charles xavier'), (1, 'bobby drake'), (2, 'kurt wagner'), (3, 'max
eisenhardt'), (4, 'kitty pryde')]
0 charles xavier
1 bobby drake
2 kurt wagner
3 max eisenhardt
4 kitty pryde
1 charles xavier
2 bobby drake
```

```
3 kurt wagner
4 max eisenhardt
5 kitty pryde
```

[946]:
```
mutants = ['charles xavier', 'bobby drake', 'kurt wagner', 'max eisenhardt',␣
 ↪'kitty pryde']
aliases = ['prof x', 'iceman', 'nightcrawler', 'magneto', 'shadowcat']
powers = ['telepathy', 'thermokinesis', 'teleportation', 'magnetokinesis',␣
 ↪'intangibility']

mutant_data = list(zip(mutants, aliases, powers))

print(mutant_data)

mutant_zip = zip(mutants, aliases, powers)

print(mutant_zip)

for value1, value2, value3 in mutant_zip:
    print(value1, value2, value3)
```

```
[('charles xavier', 'prof x', 'telepathy'), ('bobby drake', 'iceman',
'thermokinesis'), ('kurt wagner', 'nightcrawler', 'teleportation'), ('max
eisenhardt', 'magneto', 'magnetokinesis'), ('kitty pryde', 'shadowcat',
'intangibility')]
<zip object at 0x000001A95CA57700>
charles xavier prof x telepathy
bobby drake iceman thermokinesis
kurt wagner nightcrawler teleportation
max eisenhardt magneto magnetokinesis
kitty pryde shadowcat intangibility
```

4.1.1 Iteradores para cargar archivos muy grandes

[947]:
```
# Initialize an empty dictionary: counts_dict
counts_dict = {}

# Iterate over the file chunk by chunk
for chunk in pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/tweets.csv",␣
 ↪chunksize = 10):

    # Iterate over the column in DataFrame
    for entry in chunk["lang"]:
        if entry in counts_dict.keys():
            counts_dict[entry] += 1
        else:
            counts_dict[entry] = 1
```

```
# Print the populated dictionary
print(counts_dict)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

[948]:
```
counts_dict = {}

# Iterate over the file chunk by chunk
for chunk in pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/tweets.csv",␣
 ↪chunksize = 10):

    # Iterate over the column in DataFrame
    for entry in chunk["lang"]:
        if entry in counts_dict.keys():
            counts_dict[entry] += 1
        else:
            counts_dict[entry] = 1

# Print the populated dictionary
print(counts_dict)
```

```
{'en': 97, 'et': 1, 'und': 2}
```

[949]:
```
def count_entries(csv_file, c_size, colname):
    """Return a dictionary with counts of
    occurrences as value for each key."""

    counts_dict = {}

    for chunk in pd.read_csv(csv_file, chunksize = c_size):

        for entry in chunk[colname]:
            if entry in counts_dict.keys():
                counts_dict[entry] += 1
            else:
                counts_dict[entry] = 1

    return counts_dict

result_counts = count_entries("C:/Users/marco/Data Camp Python/Datasets/tweets.
 ↪csv", 10, "lang")
result_counts1 = count_entries("C:/Users/marco/Data Camp Python/Datasets/tweets.
 ↪csv", 10, "retweet_count")
result_counts2 = count_entries("C:/Users/marco/Data Camp Python/Datasets/tweets.
 ↪csv", 10, "possibly_sensitive")
print(result_counts)
print(result_counts1)
print(result_counts2)
```

```
{'en': 97, 'et': 1, 'und': 2}
{0: 100}
{False: 70, nan: 28, True: 2}
```

4.2 Listas de comprensión y generadores

[950]:
```
nums = [12, 8, 21, 3, 16]
new_nums = [num + 1 for num in nums]
print(new_nums)

# Son útiles para colapsar bucles para armar listas en una sola línea.
# Sus componentes son (1) un iterable, (2) una variable iteradora, y (3) una␣
 ↪expresión de salida
```

```
[13, 9, 22, 4, 17]
```

[951]:
```
doctor = ['house', 'cuddy', 'chase', 'thirteen', 'wilson']
[doc[0] for doc in doctor]
```

[951]:
```
['h', 'c', 'c', 't', 'w']
```

[952]:
```
squares = [i*i for i in range(10)]

print(squares)

# Se puede crear una matriz:

matrix = [[col for col in range(5)] for row in range(5)]

for row in matrix:
    print(row)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
```

4.2.1 Condiciionales en comprensiones

[953]:
```
[num ** 2 for num in range(11) if num % 2 == 0]
```

[953]:
```
[0, 4, 16, 36, 64, 100]
```

[954]:
```
fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir',␣
 ↪'gimli']

new_fellowship = [member for member in fellowship if len(member) >= 7]
```

```
print(new_fellowship)
```

```
['samwise', 'aragorn', 'legolas', 'boromir']
```

[955]:
```
fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir',␣
 ↪'gimli']

new_fellowship = [member if len(member) >= 7 else "" for member in fellowship]

print(new_fellowship)
```

```
['', 'samwise', '', 'aragorn', 'legolas', 'boromir', '']
```

[956]:
```
# Creando un diccionario:

fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir',␣
 ↪'gimli']

new_fellowship = {member: len(member) for member in fellowship}

print(new_fellowship)
```

```
{'frodo': 5, 'samwise': 7, 'merry': 5, 'aragorn': 7, 'legolas': 7, 'boromir': 7,
'gimli': 5}
```

4.2.2 Expresiones generadoras

Un generador es como una lista de compresión excepto que no almacena la lista en la memoria: no construye una lista, pero es un objeto sobre el que podemos iterar para producir elementos de la lista según sea necesario.

[957]:
```
result = (num for num in range(31))

print(next(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result))

for value in result:
    print(value)
```

```
0
1
2
3
4
5
6
```

```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

[958]:
```
lannister = ['cersei', 'jaime', 'tywin', 'tyrion', 'joffrey']

lengths = (len(person) for person in lannister)

for value in lengths:
    print(value)
```

```
6
5
5
6
7
```

[959]:
```
lannister = ['cersei', 'jaime', 'tywin', 'tyrion', 'joffrey']

def get_lengths(input_list):
    """Generator function that yields the
    length of the strings in input_list."""

    for person in input_list:
        yield len(person)

for value in get_lengths(lannister):
```

```
    print(value)
```

```
6
5
5
6
7
```

[960]:
```
# Ejemplo

# Para extraer la hora de un tuit:

tweet_time = twitter["created_at"]

tweet_clock_time = [entry[11:19] for entry in tweet_time]

print(tweet_clock_time)
```

```
['23:40:17', '23:40:17', '23:40:17', '23:40:17', '23:40:17', '23:40:17',
'23:40:18', '23:40:17', '23:40:18', '23:40:18', '23:40:18', '23:40:17',
'23:40:18', '23:40:18', '23:40:17', '23:40:18', '23:40:18', '23:40:17',
'23:40:18', '23:40:17', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:17', '23:40:18', '23:40:18', '23:40:17', '23:40:18', '23:40:18',
'23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18', '23:40:18',
'23:40:19', '23:40:18', '23:40:18', '23:40:18', '23:40:19', '23:40:19',
'23:40:19', '23:40:18', '23:40:19', '23:40:19', '23:40:19', '23:40:18',
'23:40:19', '23:40:19', '23:40:19', '23:40:18', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19']
```

[961]:
```
tweet_time = twitter["created_at"]

tweet_clock_time = [entry[11:19] for entry in tweet_time if entry[17:19] ==␣
 ↪"19"]

print(tweet_clock_time)
```

```
['23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19', '23:40:19',
'23:40:19', '23:40:19', '23:40:19', '23:40:19']
```

4.3 Proyecto integrador

[962]:
```
world_bank = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/
 ↪world_ind_pop_data.csv")

feature_names = ['CountryName', 'CountryCode', 'IndicatorName',␣
 ↪'IndicatorCode', 'Year', 'Value']
row_vals = ['Arab World', 'ARB', 'Adolescent fertility rate (births per 1,000␣
 ↪women ages 15-19)', 'SP.ADO.TFRT', '1960', '133.56090740552298']

# Zip lists: zipped_lists
zipped_lists = zip(feature_names, row_vals)

# Create a dictionary: rs_dict
rs_dict = dict(zipped_lists)

# Print the dictionary
print(rs_dict)
```

```
{'CountryName': 'Arab World', 'CountryCode': 'ARB', 'IndicatorName': 'Adolescent
fertility rate (births per 1,000 women ages 15-19)', 'IndicatorCode':
'SP.ADO.TFRT', 'Year': '1960', 'Value': '133.56090740552298'}
```

[963]:
```
# Para contar el número de veces que un país aparece en una columna:

# Open a connection to the file
with open("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv") as␣
 ↪file:

    # Skip the column names
    file.readline()

    # Initialize an empty dictionary: counts_dict
    counts_dict = {}

    # Process only the first 1000 rows
    for j in range(1000):

        # Split the current line into a list: line
        line = file.readline().split(',')

        # Get the value for the first column: first_col
        first_col = line[0]

        # If the column value is in the dict, increment its value
        if first_col in counts_dict.keys():
            counts_dict[first_col] += 1
```

```
        # Else, add to the dict and set value to 1
        else:
            counts_dict[first_col] = 1

# Print the resulting dictionary
print(counts_dict)
```

```
{'Arab World': 5, 'Caribbean small states': 5, 'Central Europe and the Baltics':
5, 'East Asia & Pacific (all income levels)': 5, 'East Asia & Pacific
(developing only)': 5, 'Euro area': 5, 'Europe & Central Asia (all income
levels)': 5, 'Europe & Central Asia (developing only)': 5, 'European Union': 5,
'Fragile and conflict affected situations': 5, 'Heavily indebted poor countries
(HIPC)': 5, 'High income': 5, 'High income: nonOECD': 5, 'High income: OECD': 5,
'Latin America & Caribbean (all income levels)': 5, 'Latin America & Caribbean
(developing only)': 5, 'Least developed countries: UN classification': 5, 'Low &
middle income': 5, 'Low income': 5, 'Lower middle income': 5, 'Middle East &
North Africa (all income levels)': 5, 'Middle East & North Africa (developing
only)': 5, 'Middle income': 5, 'North America': 5, 'OECD members': 5, 'Other
small states': 5, 'Pacific island small states': 5, 'Small states': 5, 'South
Asia': 5, 'Sub-Saharan Africa (all income levels)': 5, 'Sub-Saharan Africa
(developing only)': 5, 'Upper middle income': 5, 'World': 4, 'Afghanistan': 4,
'Albania': 4, 'Algeria': 4, 'American Samoa': 4, 'Andorra': 4, 'Angola': 4,
'Antigua and Barbuda': 4, 'Argentina': 4, 'Armenia': 4, 'Aruba': 4, 'Australia':
4, 'Austria': 4, 'Azerbaijan': 4, '"Bahamas': 4, 'Bahrain': 4, 'Bangladesh': 4,
'Barbados': 4, 'Belarus': 4, 'Belgium': 4, 'Belize': 4, 'Benin': 4, 'Bermuda':
4, 'Bhutan': 4, 'Bolivia': 4, 'Bosnia and Herzegovina': 4, 'Botswana': 4,
'Brazil': 4, 'Brunei Darussalam': 4, 'Bulgaria': 4, 'Burkina Faso': 4,
'Burundi': 4, 'Cabo Verde': 4, 'Cambodia': 4, 'Cameroon': 4, 'Canada': 4,
'Cayman Islands': 4, 'Central African Republic': 4, 'Chad': 4, 'Channel
Islands': 4, 'Chile': 4, 'China': 4, 'Colombia': 4, 'Comoros': 4, '"Congo': 8,
'Costa Rica': 4, "Cote d'Ivoire": 4, 'Croatia': 4, 'Cuba': 4, 'Curacao': 4,
'Cyprus': 4, 'Czech Republic': 4, 'Denmark': 4, 'Djibouti': 4, 'Dominica': 4,
'Dominican Republic': 4, 'Ecuador': 4, '"Egypt': 4, 'El Salvador': 4,
'Equatorial Guinea': 4, 'Eritrea': 4, 'Estonia': 4, 'Ethiopia': 4, 'Faeroe
Islands': 4, 'Fiji': 4, 'Finland': 4, 'France': 4, 'French Polynesia': 4,
'Gabon': 4, '"Gambia': 4, 'Georgia': 4, 'Germany': 4, 'Ghana': 4, 'Greece': 4,
'Greenland': 4, 'Grenada': 4, 'Guam': 4, 'Guatemala': 4, 'Guinea': 4, 'Guinea-
Bissau': 4, 'Guyana': 4, 'Haiti': 4, 'Honduras': 4, '"Hong Kong SAR': 4,
'Hungary': 4, 'Iceland': 4, 'India': 4, 'Indonesia': 4, '"Iran': 4, 'Iraq': 4,
'Ireland': 4, 'Isle of Man': 4, 'Israel': 4, 'Italy': 4, 'Jamaica': 4, 'Japan':
4, 'Jordan': 4, 'Kazakhstan': 4, 'Kenya': 4, 'Kiribati': 4, '"Korea': 8,
'Kuwait': 4, 'Kyrgyz Republic': 4, 'Lao PDR': 4, 'Latvia': 4, 'Lebanon': 4,
'Lesotho': 4, 'Liberia': 4, 'Libya': 4, 'Liechtenstein': 4, 'Lithuania': 4,
'Luxembourg': 4, '"Macao SAR': 4, '"Macedonia': 4, 'Madagascar': 4, 'Malawi': 4,
'Malaysia': 4, 'Maldives': 4, 'Mali': 4, 'Malta': 4, 'Marshall Islands': 4,
'Mauritania': 4, 'Mauritius': 4, 'Mexico': 4, '"Micronesia': 4, 'Moldova': 4,
'Monaco': 4, 'Mongolia': 4, 'Montenegro': 4, 'Morocco': 4, 'Mozambique': 4,
'Myanmar': 4, 'Namibia': 4, 'Nepal': 4, 'Netherlands': 4, 'New Caledonia': 4,
```

```
'New Zealand': 4, 'Nicaragua': 4, 'Niger': 4, 'Nigeria': 4, 'Northern Mariana
Islands': 4, 'Norway': 4, 'Oman': 4, 'Pakistan': 4, 'Palau': 4, 'Panama': 4,
'Papua New Guinea': 4, 'Paraguay': 4, 'Peru': 4, 'Philippines': 4, 'Poland': 4,
'Portugal': 4, 'Puerto Rico': 4, 'Qatar': 4, 'Romania': 4, 'Russian Federation':
4, 'Rwanda': 4, 'Samoa': 4, 'San Marino': 4, 'Sao Tome and Principe': 4, 'Saudi
Arabia': 4, 'Senegal': 4, 'Seychelles': 4, 'Sierra Leone': 4, 'Singapore': 4,
'Slovak Republic': 4, 'Slovenia': 4, 'Solomon Islands': 4, 'Somalia': 4, 'South
Africa': 4, 'South Sudan': 4, 'Spain': 4, 'Sri Lanka': 4, 'St. Kitts and Nevis':
4, 'St. Lucia': 4, 'St. Vincent and the Grenadines': 4, 'Sudan': 4, 'Suriname':
4, 'Swaziland': 4, 'Sweden': 4, 'Switzerland': 4, 'Syrian Arab Republic': 4,
'Tajikistan': 4, 'Tanzania': 4, 'Thailand': 4, 'Timor-Leste': 4, 'Togo': 4,
'Tonga': 4, 'Trinidad and Tobago': 4, 'Tunisia': 4, 'Turkey': 4, 'Turkmenistan':
4, 'Turks and Caicos Islands': 4, 'Tuvalu': 4, 'Uganda': 4, 'Ukraine': 4,
'United Arab Emirates': 4, 'United Kingdom': 4, 'United States': 4, 'Uruguay':
4, 'Uzbekistan': 4, 'Vanuatu': 4, '"Venezuela': 4, 'Vietnam': 4, 'Virgin Islands
(U.S.)': 4, '"Yemen': 4, 'Zambia': 4, 'Zimbabwe': 4}
```

[964]:
```
# Para cargar los datos en chunks:

# Define read_large_file()
def read_large_file(file_object):
    """A generator function to read a large file lazily."""

    # Loop indefinitely until the end of the file
    while True:

        # Read a line from the file: data
        data = file_object.readline()

        # Break if this is the end of the file
        if not data:
            break

        # Yield the line of data
        yield data

# Open a connection to the file
with open('C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv') as␣
 ↪file:

    # Create a generator object for the file: gen_file
    gen_file = read_large_file(file)

    # Print the first fice lines of the file
    print(next(gen_file))
    print(next(gen_file))
    print(next(gen_file))
```

```
    print(next(gen_file))
    print(next(gen_file))
```

```
CountryName,CountryCode,Year,Total Population,Urban population (% of total)

Arab World,ARB,1960,92495902.0,31.285384211605397

Caribbean small states,CSS,1960,4190810.0,31.5974898513652

Central Europe and the Baltics,CEB,1960,91401583.0,44.5079211390026

East Asia & Pacific (all income levels),EAS,1960,1042475394.0,22.471132204295397
```

[965]:
```
# pd.read_csv como iterador

import pandas as pd

df_reader = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/
 ↪world_ind_pop_data.csv", chunksize = 10)

print(next(df_reader))
```

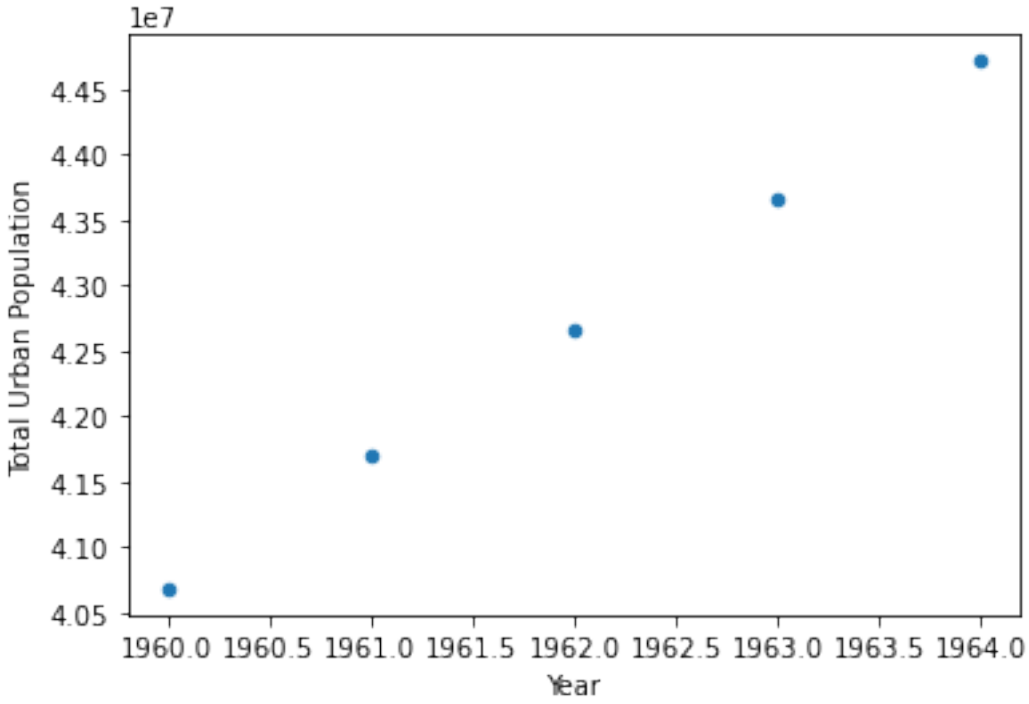
```
                               CountryName CountryCode  Year  \
0                               Arab World         ARB  1960
1                     Caribbean small states         CSS  1960
2             Central Europe and the Baltics         CEB  1960
3    East Asia & Pacific (all income levels)         EAS  1960
4      East Asia & Pacific (developing only)         EAP  1960
5                                 Euro area         EMU  1960
6  Europe & Central Asia (all income levels)         ECS  1960
7    Europe & Central Asia (developing only)         ECA  1960
8                             European Union         EUU  1960
9   Fragile and conflict affected situations         FCS  1960

   Total Population  Urban population (% of total)
0      9.249590e+07                      31.285384
1      4.190810e+06                      31.597490
2      9.140158e+07                      44.507921
3      1.042475e+09                      22.471132
4      8.964930e+08                      16.917679
5      2.653965e+08                      62.096947
6      6.674890e+08                      55.378977
7      1.553174e+08                      38.066129
8      4.094985e+08                      61.212898
9      1.203546e+08                      17.891972
```

[966]:
```
# Initialize reader object: urb_pop_reader
urb_pop_reader = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/
 ↪world_ind_pop_data.csv", chunksize = 1000)

# Get the first DataFrame chunk: df_urb_pop
df_urb_pop = next(urb_pop_reader)

# Check out the head of the DataFrame
print(df_urb_pop.head())

# Check out specific country: df_pop_ceb
df_pop_ceb = df_urb_pop[df_urb_pop["CountryCode"] == "CEB"]

# Zip DataFrame columns of interest: pops
pops = zip(df_pop_ceb["Total Population"], df_pop_ceb["Urban population (% of␣
 ↪total)"])

# Turn zip object into list: pops_list
pops_list = list(pops)

# Print pops_list
print(pops_list)
```

```
                            CountryName CountryCode  Year  \
0                            Arab World         ARB  1960
1                  Caribbean small states         CSS  1960
2           Central Europe and the Baltics         CEB  1960
3  East Asia & Pacific (all income levels)         EAS  1960
4    East Asia & Pacific (developing only)         EAP  1960

   Total Population  Urban population (% of total)
0      9.249590e+07                      31.285384
1      4.190810e+06                      31.597490
2      9.140158e+07                      44.507921
3      1.042475e+09                      22.471132
4      8.964930e+08                      16.917679
[(91401583.0, 44.5079211390026), (92237118.0, 45.206665319194), (93014890.0,
45.866564696018), (93845749.0, 46.5340927663649), (94722599.0,
47.2087429803526)]
```

[967]:
```
# Use list comprehension to create new DataFrame column 'Total Urban Population'
df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for tup in␣
 ↪pops_list]

# Plot urban population data
df_pop_ceb.plot(kind="scatter", x="Year", y="Total Urban Population")
plt.show()
```

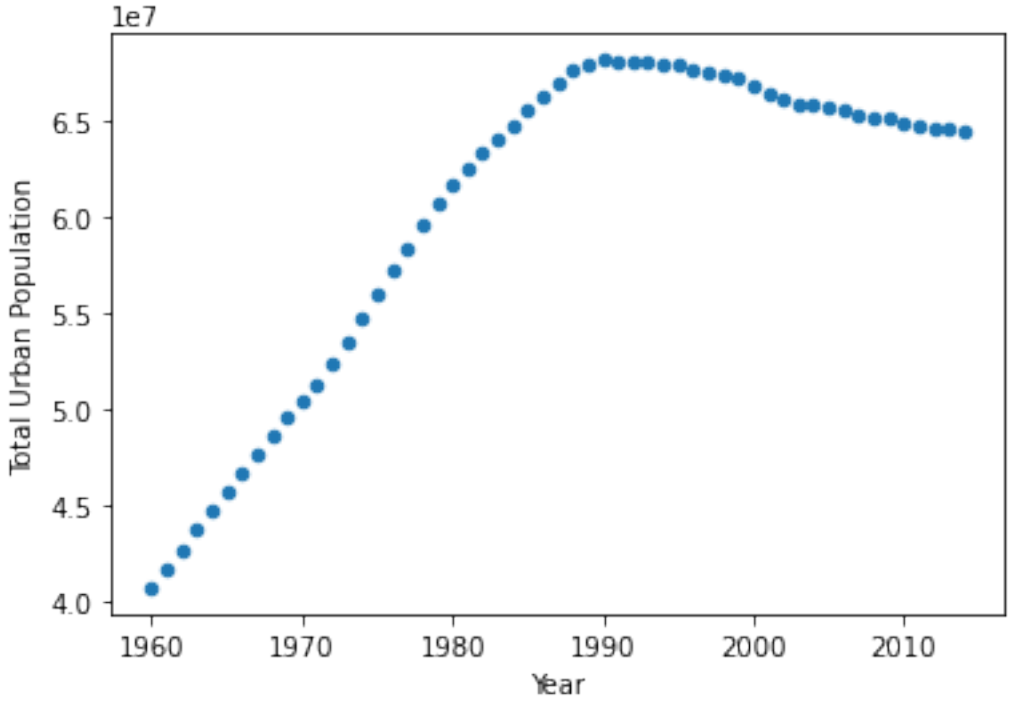
```
<ipython-input-967-906002f22436>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for tup in
pops_list]
```



[968]:
```
# Initialize reader object: urb_pop_reader
urb_pop_reader = pd.read_csv('C:/Users/marco/Data Camp Python/Datasets/
 ↪world_ind_pop_data.csv', chunksize=1000)

# Initialize empty DataFrame: data
data = pd.DataFrame()

# Iterate over each DataFrame chunk
for df_urb_pop in urb_pop_reader:

    # Check out specific country: df_pop_ceb
    df_pop_ceb = df_urb_pop[df_urb_pop['CountryCode'] == 'CEB']

    # Zip DataFrame columns of interest: pops
    pops = zip(df_pop_ceb['Total Population'],
```

```
                df_pop_ceb['Urban population (% of total)'])

    # Turn zip object into list: pops_list
    pops_list = list(pops)

    # Use list comprehension to create new DataFrame column 'Total Urban␣
↪Population'
    df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for tup␣
↪in pops_list]

    # Append DataFrame chunk to data: data
    data = data.append(df_pop_ceb)

# Plot urban population data
data.plot(kind='scatter', x='Year', y='Total Urban Population')
plt.show()
```

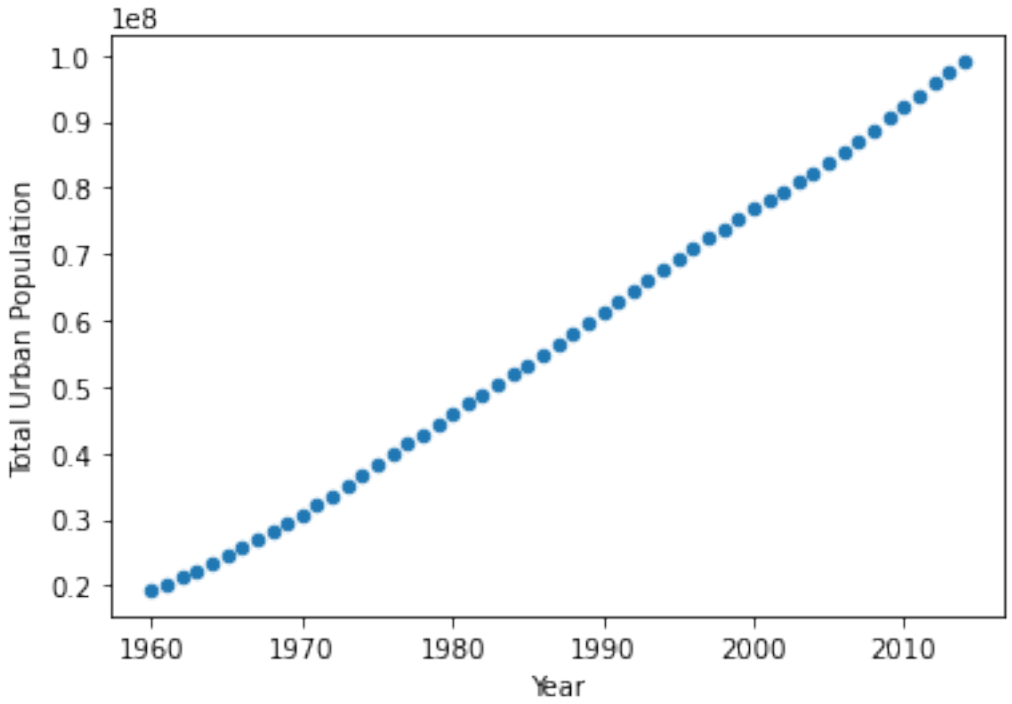
```
<ipython-input-968-2e590769ff6b>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for tup in
pops_list]
```

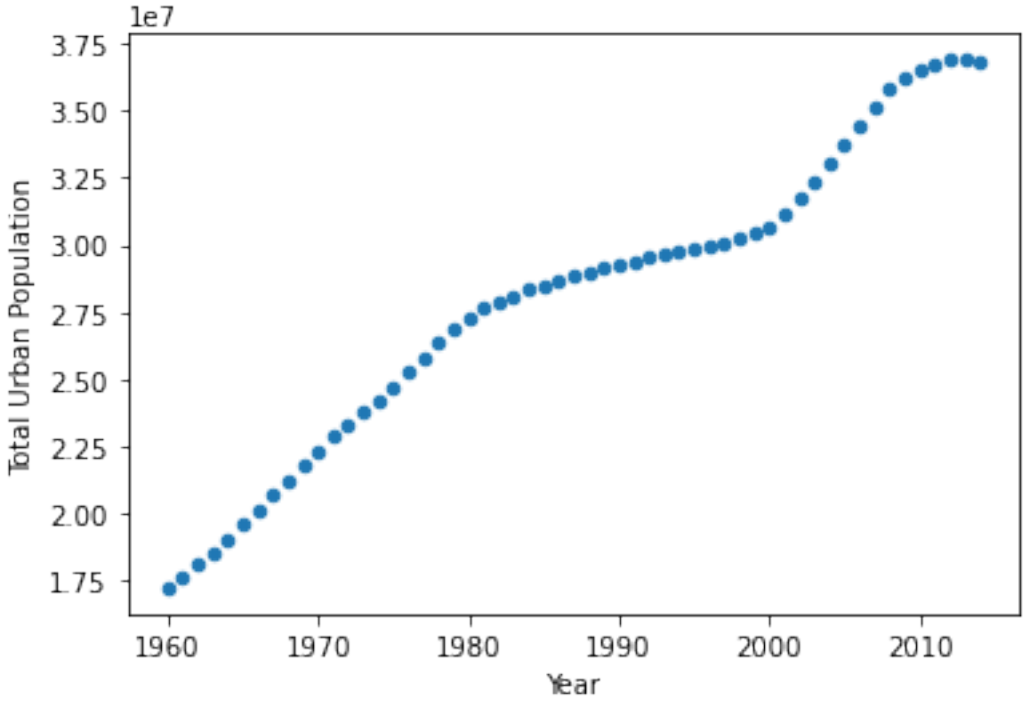


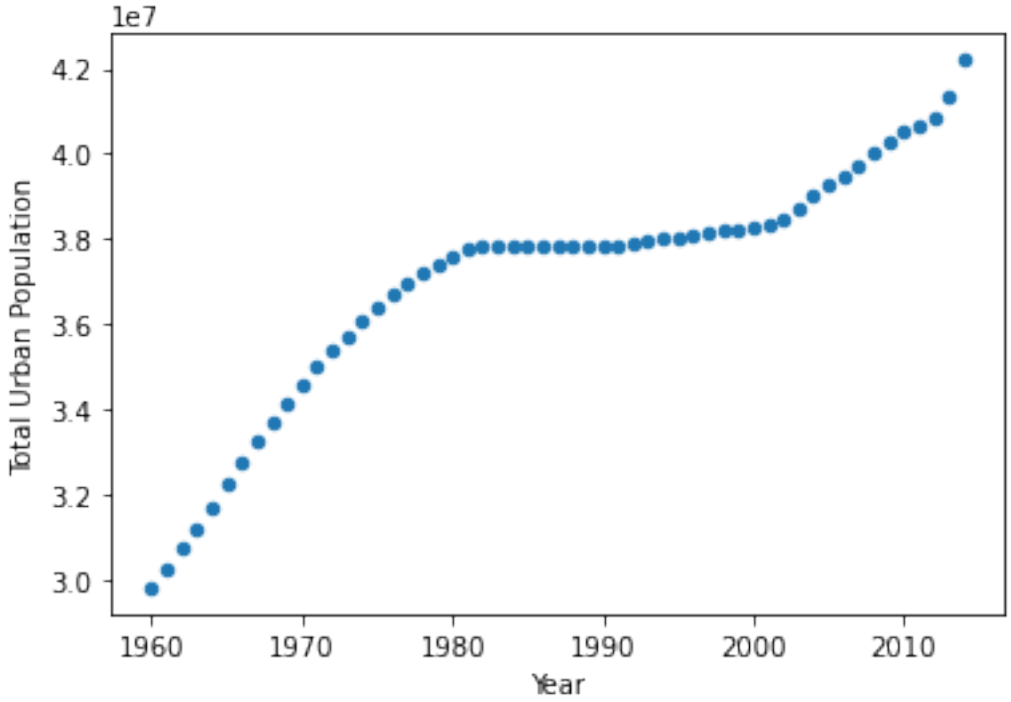
[969]:
```
# Para generalizar la graficación a partir del dataset:

# Define plot_pop()
def plot_pop(filename, country_code):

    # Initialize reader object: urb_pop_reader
    urb_pop_reader = pd.read_csv(filename, chunksize=1000)

    # Initialize empty DataFrame: data
    data = pd.DataFrame()

    # Iterate over each DataFrame chunk
    for df_urb_pop in urb_pop_reader:
        # Check out specific country: df_pop_ceb
        df_pop_ceb = df_urb_pop[df_urb_pop['CountryCode'] == country_code]

        # Zip DataFrame columns of interest: pops
        pops = zip(df_pop_ceb['Total Population'],
                   df_pop_ceb['Urban population (% of total)'])

        # Turn zip object into list: pops_list
        pops_list = list(pops)

        # Use list comprehension to create new DataFrame column 'Total Urban␣
 ↪Population'
        df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for␣
 ↪tup in pops_list]

        # Append DataFrame chunk to data: data
        data = data.append(df_pop_ceb)

    # Plot urban population data
    data.plot(kind='scatter', x='Year', y='Total Urban Population')
    plt.show()

# Set the filename: fn
fn = 'C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv'


plot_pop("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv",␣
 ↪"MEX")
plot_pop("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv",␣
 ↪"ESP")
plot_pop("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv",␣
 ↪"ITA")
```

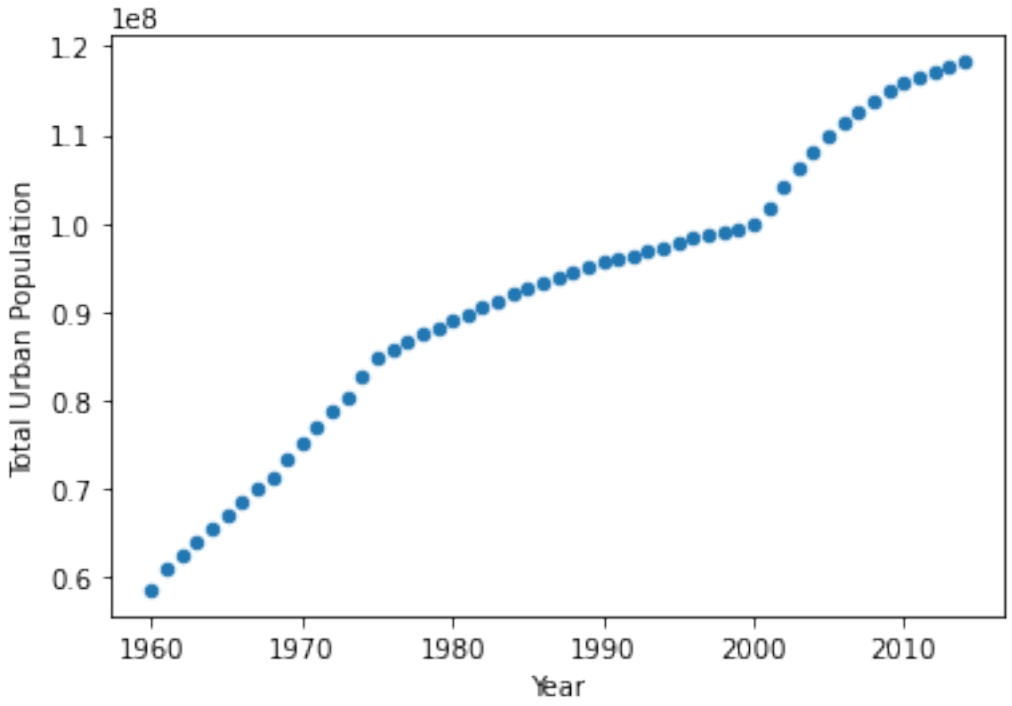
```
plot_pop("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv",␣
 ↪"JPN")
plot_pop("C:/Users/marco/Data Camp Python/Datasets/world_ind_pop_data.csv",␣
 ↪"KOR")
```

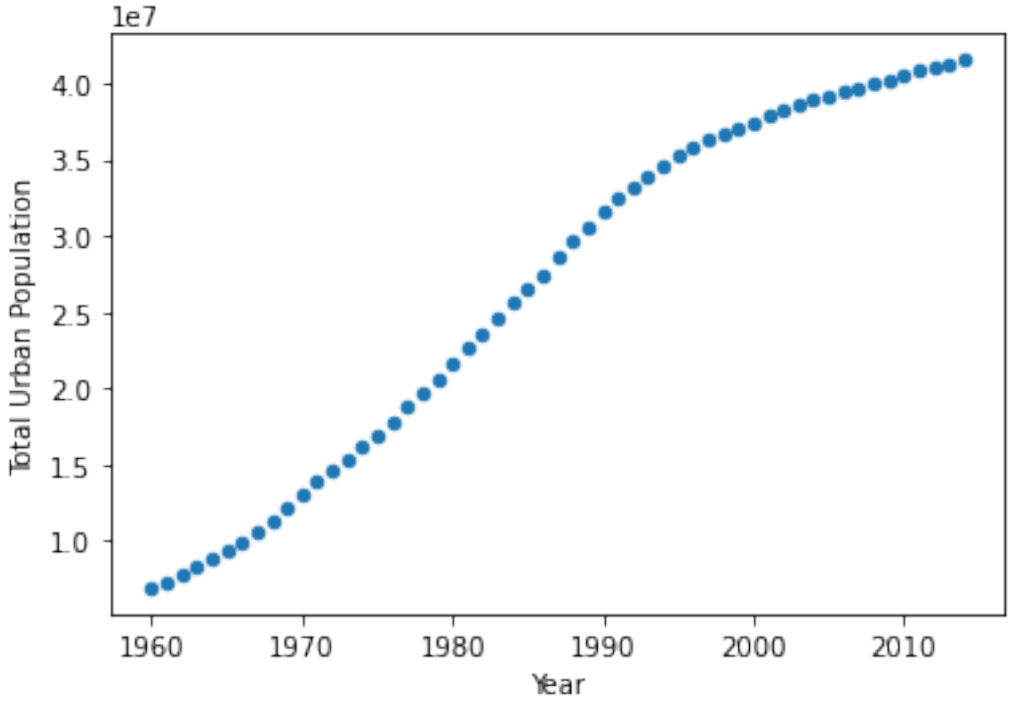
```
<ipython-input-969-722b39193383>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_pop_ceb['Total Urban Population'] = [int(tup[0] * tup[1] * 0.01) for tup in
pops_list]
```




1e7
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
Total Urban Population
1960
1970
1980
1990
2000
2010
Year


1e7
4.2
4.0
3.8
3.6
3.4
3.2
3.0
Total Urban Population
1960
1970
1980
1990
2000
2010
Year


1e8
1.2
1.1
1.0
0.9
0.8
0.7
0.6
Total Urban Population
1960
1970
1980
1990
2000
2010
Year


1e7
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Total Urban Population
1960
1970
1980
1990
2000
2010
Year