Ejercicios de introducción a Python

1.- Escribe un código que obtenga la longitud de las cadenas de tamaño mayor que 2 que comiezan y acaban por el mismo caracter:

```
In: words = ['abc', 'xyz', 'aba', '1221', 'aa']
Out: [3, 4]
```

In [1]:
```
words = ['abc', 'xyz', 'aba', '1221', 'aa']
'''
cadenas = []
for word in words:
    if len(word)>2 and word[0] == word[-1]:
        cadenas.append(len(word))
cadenas
'''
# list(map(len, filter(lambda word: len(word)>2 and word[0] == word[
[len(word) for word in words if len(word)>2 and word[0] == word[-1]]
```

Out[1]:
```
[3, 4]
```

2.- Escribir un código que dada una lista de tuplas no vacias, devuelva una lista ordenada por el último elemento de las tuplas en orden creciente:

```
In: [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]
Out: [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]
```

In [2]:
```
tuple_list = [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

def por_posicion (x, posicion):
    return x[posicion]

def por_final (x):
    return x[-1]

# sorted devuelve una nueva lista ordenada
sorted(tuple_list, key=(lambda x: por_posicion(x, -1)))
sorted(tuple_list, key=por_final)
sorted(tuple_list, key=lambda x: x[-1])
# sort modifica la lista
# tuple_list.sort(key=(lambda x: por_posicion(x, -1)))
# tuple_list.sort(key=por_final)
# tuple_list.sort(key=lambda x: x[-1])
```

Out[2]:
```
[(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]
```

3.- Escribir un código que devuelva una lista resultado de eliminar los dupliados de otra:

```
In: duplicates = [10,20,30,20,10,50,60,40,80,50,40]
Out: [40, 10, 80, 50, 20, 60, 30]
```

In [3]:
```
duplicates = [10,20,30,20,10,50,60,40,80,50,40]
'''
out = []
for valor in duplicates:
    if valor not in out:
        out.append(valor)
out
'''
list(set(duplicates))
```

Out[3]: [40, 10, 80, 50, 20, 60, 30]

4.- Escribir un código que devuelve si dadas dos listas comparten algún elemento:

```
In: list1 = [1,2,3,4,5], list2 = [5,6,7,8,9]
Out: True
```

In [4]:
```
list1 = [1,2,3,4,5]
list2 = [5,6,7,8,9]
'''
comparten = False
for value1 in list1:
    for value2 in list2:
        if value1 == value2:
            comparten = True
            break
    if comparten:
        break

print(comparten)
'''
# print(any(value1 == value2 for value1 in list1 for value2 in list2
'''
comparten = False
for value1 in list1:
    if value1 in list2:
        comparten = True
        break
print(comparten)
'''
# print(any(value1 in list2 for value1 in list1))
# True in [value1 in list2 for value1 in list1]
# bool(set(list1).intersection(set(list2)))
bool(set(list1) & set(list2))
```

Out[4]: True

4.1.- Devolver los elementos comunes.

In [5]:
```
# set(list1).intersection(set(list2))
# set(list1) & set(list2)
list(filter(lambda x: x in list1, list2))
```

Out[5]: [5]

5.- Escribir un código que elimine los valores impares de una lista:

```
In: list1 = [1,2,3,4,5]
Out: [2, 4]
```

In [6]:
```
list1 = [1,2,3,4,5]
[value for value in list1 if value%2==0]
# list(filter(lambda value: value%2==0, list1))
```

Out[6]:
```
[2, 4]
```

6.- Escribir un código que devuelva una lista con los elementos diferentes de otras dos listas:

```
In: list1 = [1,2,3,4,5], list2 = [5,6,7,8,9]
Out: [1, 2, 3, 4, 6, 7, 8, 9]
```

In [7]:
```
list1 = [1,2,3,4,5]
list2 = [5,6,7,8,9]
'''
out = []
for value in list1+list2:
    if not (value in list1 and value in list2):
        out.append(value)
out
'''
# [value for value in list1+list2 if not (value in list1 and value i
# list(set(list1).symmetric_difference(set(list2)))
list(set(list1) ^ set(list2))
```

Out[7]:
```
[1, 2, 3, 4, 6, 7, 8, 9]
```

7.- Escribir un código que devuelva el segundo elemento mayor de una lista:

```
In: list1 = [9,2,1,4,8,6,7,5,3]
Out: 8
```

In [8]:
```
list1 = [9,2,1,4,8,6,7,5,3]
sorted(list1)[-2]
# sorted(list1, reverse=True)[1]
```

Out[8]:
```
8
```

8.- Escribir un código que devuelva una lista ordenada con las frecuencias de los elementos de mayor a menor:

```
In: list1 = [10,10,10,10,20,20,20,20,40,40,50,50,30]
Out: [(10, 4), (20, 4), (40, 2), (50, 2), (30, 1)]
```

In [9]:
```
list1 = [10,10,10,10,20,20,20,20,40,40,50,50,30]
'''
res = []
for valor in set(list1):
    res.append((valor, list1.count(valor)))
sorted(res, key=lambda x: x[-1], reverse=True)
'''
sorted([(value, list1.count(value)) for value in set(list1)], key=la
```

Out[9]:
```
[(10, 4), (20, 4), (40, 2), (50, 2), (30, 1)]
```

9.- Escribir un código que devuelva una lista resultado de concatenar los valores de una lista con un rango:

```
In: list1 = ['p', 'q'], n = 4
Out: ['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4']
```

In [10]:
```
list1 = ['p', 'q']
n = 4
'''
out = []
for i in range(1, n+1):
    for letter in list1:
        out.append(letter+str(i))
out
'''
[letter+str(i) for i in range(1, n+1) for letter in list1]
```

Out[10]:
```
['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4']
```

10.- Escribir un código que convierta una lista de listas en un diccionario:

```
In: list1 = [["Black", "Red", "Maroon", "Yellow"],
["#000000", "#FF0000", "#800000", "#FFFF00"]]
Out: {'Black': '#FFFF00', 'Red': '#FFFF00', 'Maroon':
'#FFFF00', 'Yellow': '#FFFF00'}
```

In [11]:
```
list1 = [["Black", "Red", "Maroon", "Yellow"], ["#000000", "#FF0000"
'''
out = {}
for i, clave in enumerate(list1[0]):
    out[clave] = list1[1][i]
out
'''
'''
out = {}
for clave, valor in zip(*list1):
    out[clave] = valor
out
'''
# {clave: valor for clave, valor in zip(*list1)}
dict(zip(*list1))
```

Out[11]:
```
{'Black': '#000000',
 'Red': '#FF0000',
 'Maroon': '#800000',
```

```
 'Yellow': '#FFFF00'}
```

11.- Escribir un código que divida una lista en una lista de listas cada n elementos (no contiguos):

```
In: list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n']
     step = 3
Out: [['a', 'd', 'g', 'j', 'm'], ['b', 'e', 'h', 'k', 'n'],
['c', 'f', 'i', 'l']]
```

In [12]:

```
list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
step = 3
[list1[i::step] for i in range(step)]
```

Out[12]:

```
[['a', 'd', 'g', 'j', 'm'], ['b', 'e', 'h', 'k', 'n'], ['c', 'f', 'i
', 'l']]
```

12.- Escribir un código que divida una lista en una lista de listas cada n elementos (contiguos):

```
In: list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n']
     step = 3
Out: [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i'],
['j', 'k', 'l'], ['m', 'n']]
```

In [13]:

```
[list1[i:i+step] for i in range(0,len(list1),step)]
```

Out[13]:

```
[['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', 'i'],
 ['j', 'k', 'l'],
 ['m', 'n']]
```

13.- Escribir un código que encuentre en una lista de listas aquella cuya suma sea mayor:

```
In: list1 = [[1,2,3], [4,5,6], [10,11,12], [7,8,9]]
Out: [10,11,12]
```

In [14]:

```
list1 = [[1,2,3], [4,5,6], [10,11,12], [7,8,9]]
'''
max_list = None
sum_max = 0
for sublist in list1:
    if sum(sublist) > sum_max:
        sum_max = sum(sublist)
        max_list = sublist
max_list
'''
# list1[list(map(sum, list1)).index(max(map(sum, list1)))]
# sorted([(sublist, sum(sublist)) for sublist in list1], key=lambda
# sorted([(sublist, sum(sublist)) for sublist in list1], key=lambda
# sorted([(sublist, sum(sublist)) for sublist in list1], key=lambda
# sorted(list1, key=sum)[-1]
max(list1, key=sum)
```

Out[14]:
```
[10, 11, 12]
```

14.- Escribir un código que elimine las listas duplicadas de una lista de listas:

```
In: list1 = [[10, 20], [40], [30, 56, 25], [10, 20], [33], [40]]
Out: [[33], [30, 56, 25], [40], [10, 20]]
```

In [15]:
```
list1 = [[10, 20], [40], [30, 56, 25], [10, 20], [33], [40]]
'''
result = []
for inner_list in list1:
    if inner_list not in result:
        result.append(inner_list)
result
'''
# [list(inner_tuple) for inner_tuple in {tuple(inner_list) for inner
# [list(inner_tuple) for inner_tuple in set(tuple(inner_list) for in
# list(map(list, set(map(tuple, list1))))
import itertools
list(k for k, _ in itertools.groupby(sorted(list1)))
```

Out[15]:
```
[[10, 20], [30, 56, 25], [33], [40]]
```

15.- Escribir un código que elimine las tuplas vacias de una lista de tuplas:

```
In: list1 = [(), (), ('',), ('a', 'b'), ('a', 'b', 'c'), ('d')]
Out: [('',), ('a', 'b'), ('a', 'b', 'c'), 'd']
```

In [16]:
```
list1 = [(), (), ('',), ('a', 'b'), ('a', 'b', 'c'), ('d')]
# [lista for lista in list1 if len(lista) != 0]
# [inner_tuple for inner_tuple in list1 if inner_tuple]
list(filter(lambda x: x != tuple(), list1))
```

Out[16]:
```
[('',), ('a', 'b'), ('a', 'b', 'c'), 'd']
```

16.- Escribir un código que concatene varios diccionarios en uno:

```
In: dic1={1:10, 2:20}, dic2={3:30, 4:40}, dic3={5:50, 6:60}
Out: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

In [17]:
```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50, 6:60}
'''
dic4 = {}
for d in (dic1, dic2, dic3):
    dic4.update(d)
dic4
'''
# {1:10, 2:20, 3:30, 4:40, 5:50, 6:60}
{**dic1 , **dic2, **dic3}
```

Out[17]:
```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

17.- Escribir un código que devuelva los valores únicos de una lista de diccionarios:

```
In: list1 = [{"V":"S001", "VI": "S002"}, {"VI": "S001"},
{"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
Out: {'S009', 'S002', 'S007', 'S005', 'S001'}
```

In [18]:
```
list1 = [{"V":"S001", "VI": "S002"}, {"VI": "S001"}, {"VI": "S005"},
{values for dic in list1 for values in dic.values()}
```

Out[18]:
```
{'S001', 'S002', 'S005', 'S007', 'S009'}
```

18.- Escribir un codigo que obtenga los 3 valores más altos de un diccionario:

```
In: my_dict = {'a':500, 'b':5874, 'c': 560,'d':400, 'e':5874,
'f': 20}
Out: [5874, 5874, 560]
```

In [19]:
```
my_dict = {'a':500, 'b':5874, 'c': 560, 'd':400, 'e':5874, 'f': 20}
sorted(my_dict.values(), reverse=True)[:3]
```

Out[19]:
```
[5874, 5874, 560]
```

19.- Escribir un codigo que obtenga un diccionario a partir de una cadena en el que las letras sean las claves y los valores el número de apariciones:

```
In: str1 = 'w3resource'
Out: {'w': 1, '3': 1, 'r': 2, 'e': 2, 's': 1, 'o': 1, 'u': 1,
'c': 1}
```

In [20]:
```
str1 = 'w3resource'
'''
res = dict()
for letter in str1:
    res[letter] = str1.count(letter)
res
'''
{letter : str1.count(letter) for letter in str1}
```

Out[20]:
```
{'w': 1, '3': 1, 'r': 2, 'e': 2, 's': 1, 'o': 1, 'u': 1, 'c': 1}
```

20.- Escribir un codigo que obtenga un diccionario con el acumulado de los valores de sus claves:

```
In: list1 = [{'item': 'item1', 'amount': 400},
             {'item': 'item2', 'amount': 300},
             {'item': 'item1', 'amount': 750}]
Out: {'item1': 1150, 'item2': 300}
```

In [21]:
```
list1 = [{'item': 'item1', 'amount': 400}, {'item': 'item2', 'amount
'''
result = dict()
for inner_dict in list1:
    if inner_dict["item"] in result:
        result[inner_dict["item"]] += inner_dict["amount"]
    else:
        result[inner_dict["item"]] = inner_dict["amount"]

result
'''
result = dict()
for inner_dict in list1:
    result.setdefault(inner_dict["item"], 0)
    result[inner_dict["item"]] += inner_dict["amount"]

result
```

Out[21]:
```
{'item1': 1150, 'item2': 300}
```

Otras...

1.- Dada una lista de la compra basada en productos (producto, precio kg, kg), se pide obtener el total del coste de la misma:

```
In: lista_compra = (("patatas", 1.8, 2),
                    ("zanahorias", 2.0, 1),
                    ("aguacates", 3.5, 2))
Out: 12.6
```

In [22]:
```
lista_compra = (("patatas", 1.8, 2),
         ("zanahorias", 2.0, 1),
         ("aguacates", 3.5, 2))

'''
total = 0
for producto in lista_compra:
    total += (producto[1] * producto[2])
total
'''
sum(precio * cantidad for  _, precio, cantidad in lista_compra)
```

Out[22]:
```
12.6
```

2.- Dada una lista de la compra basada en productos (producto, precio kg, kg), se pide obtener el total del coste de la misma:

```
In: lista_compra = (("patatas", "1.8 €/Kg", "2 Kg"),
                    ("zanahorias", "2.0 €/Kg", "1 Kg"),
                    ("aguacates", "3.5 €/Kg", "2 Kg"))
Out: 12.6
```

In [23]:
```
lista_compra = (("patatas", "1.8 €/Kg", "2 Kg"),
                ("zanahorias", "2.0 €/Kg", "1 Kg"),
                ("aguacates", "3.5 €/Kg", "2 Kg"))

sum(float(precio.split()[0]) * float(cantidad.split()[0]) for _, pre
```

Out[23]: 12.6

2.1.- Utilizar una función:

In [24]:
```
def total_producto(_, precio, cantidad):
    return float(precio.split()[0]) * float(cantidad.split()[0])

sum(total_producto(*producto) for producto in lista_compra)
```

Out[24]: 12.6

2.2.- Utilizar la función **map**:

In [25]:
```
sum(map(total_producto, *zip(*lista_compra)))
```

Out[25]: 12.6

2.3.- Utilizar una función de la librería **itertoools**:

In [26]:
```
import itertools
sum(itertools.starmap(total_producto, lista_compra))
```

Out[26]: 12.6

3.- Crear una función que dadas dos cadenas nos indique si son anagramas:

```
In: is_anagram('anagram','margana')
Out: True
In: is_anagram('cat','rat')
Out: False
```

In [27]:
```
def is_anagram(str1, str2):
    return (str1 == str2[::-1])

print(is_anagram('anagram','margana'))
print(is_anagram('cat','rat'))
```

```
True
False
```

4.- Dada una lista de listas de tuplas, indicar el número de apariciones de cada una de ellas:

```
In: lista = [[('hi', 'bye')], [('Geeks', 'forGeeks')],
             [('a', 'b')], [('hi', 'bye')], [('a', 'b')]]
Out: {('hi', 'bye'): 2, ('Geeks', 'forGeeks'): 1, ('a', 'b'):
2}
```

In [28]:
```
lista = [[('hi', 'bye')], [('Geeks', 'forGeeks')],
         [('a', 'b')], [('hi', 'bye')], [('a', 'b')]]
out = dict()
for elem in lista:
    out.setdefault(elem[0], 0)
    out[elem[0]] += 1
out
```

Out[28]:
```
{('hi', 'bye'): 2, ('Geeks', 'forGeeks'): 1, ('a', 'b'): 2}
```

5.- Dada una lista de tuplas, obtener utilizando un **filtro** aquellas que continen un valor específico:

```
In: lista = [(11, 22), (33, 55), (55, 77), (11, 44), (33, 22,
100, 11), (99, 11)]
     n = 11
Out: [(11, 22), (11, 44), (33, 22, 100, 11), (99, 11)]
```

In [29]:
```
lista = [(11, 22), (33, 55), (55, 77), (11, 44), (33, 22, 100, 11),
n = 11

# [tupla for tupla in lista if n in tupla]
list(filter(lambda x: n in x, lista))
```

Out[29]:
```
[(11, 22), (11, 44), (33, 22, 100, 11), (99, 11)]
```

6.- Sumar los valores de dos listas utilizando la función **map** con una función **lambda**:

```
In: list1 = [1, 2, 3]
     list2 = [4, 5, 6]
Out: [5, 7, 9]
```

In [30]:
```
list1 = [1, 2, 3]
list2 = [4, 5, 6]

# [x + y for x, y in zip(list1, list2)]
# list(map(sum, zip(list1, list2)))
# list(map(lambda x: x[0] + x[1], zip(list1, list2)))
list(map(lambda x, y: x + y, list1, list2))
```

Out[30]:
```
[5, 7, 9]
```

7.- El siguiente diccionario contiene el ranking de varias compañías del Fortune 500:

```
# Dictionary of fortune 500 companies
    dictOfFortune500 = {
        "Walmart": 1,
        "Exxon Mobil" : 2,
        "Berkshire Hathaway" : 3,
        "Apple" : 4,
        "UnitedHealth Group" : 5,
        "McKesson" : 5,
```

```
"CVS Health" : 5,
"Amazon.com" : 6,
"AT&T" : 6,
"General Motors" : 7
}
```

In [31]:

```
# Dictionary of fortune 500 companies
dictOfFortune500 = {
    "Walmart": 1,
    "Exxon Mobil" : 2,
    "Berkshire Hathaway" : 3,
    "Apple" : 4,
    "UnitedHealth Group" : 5,
    "McKesson" : 5,
    "CVS Health" : 5,
    "Amazon.com" : 6,
    "AT&T" : 6,
    "General Motors" : 7
    }
```

7.1.- Definir una función que devuelva las compañías con ranking 5:

In [32]:

```
'''
Get the list of Companies having world ranking 5 using a function
'''

def searchKeysByVal(dict, byVal):
    keysList = []
    itemsList = dict.items()
    for item in itemsList:
        if item[1] == byVal:
            keysList.append(item[0])
    return keysList

keysList = searchKeysByVal(dictOfFortune500, 5)

print("Fortune 500 Companies having world raking '5' are:", end = "\
#Iterate over the list of companies
for index, company in enumerate(keysList):
    print("{}: {}".format(index, company))
```

```
Fortune 500 Companies having world raking '5' are:

0: UnitedHealth Group
1: McKesson
2: CVS Health
```

7.2.- Utilizar una comprensión de listas:

In [33]:
```
'''
Get the list of Companies having world ranking 5 using list comprehe
'''

keysList = [company  for (company, value) in dictOfFortune500.items(

print("Fortune 500 Companies having world raking '5' are:", end = "\
#Iterate over the list of companies
for index, company in enumerate(keysList):
    print("{}: {}".format(index, company))
```

```
Fortune 500 Companies having world raking '5' are:

0: UnitedHealth Group
1: McKesson
2: CVS Health
```

7.3.- Utilizar un filtro:

In [34]:
```
'''
Get the list of Companies having world ranking 5 using a filter
'''

out = dict(filter(lambda elem: elem[1] == 5, dictOfFortune500.items(
print("Fortune 500 Companies having world raking '5' are:", end = "\
#Iterate over the list of companies
for index, company in enumerate(out):
    print("{}: {}".format(index, company))
```

```
Fortune 500 Companies having world raking '5' are:

0: UnitedHealth Group
1: McKesson
2: CVS Health
```

7.4.- Modificar la función para que admita una lista de valores:

In [35]:
```
'''
Get the list of Companies whose rank matches with values in the inpu
'''

def searchKeysByValList(itemDict, valList):
    keysList = []
    itemsList = itemDict.items()
    for item  in itemsList:
        if item[1] in valList:
            keysList.append(item[0])
    return  keysList

'''
Get the list of Companies matching any of the input  values
'''

keysList = searchKeysByValList(dictOfFortune500, [5, 6] )

#Iterate over the list of values
for key in keysList:
    print(key)
```

```
UnitedHealth Group
McKesson
CVS Health
Amazon.com
AT&T
```

[Python] [test] [ejercicios]