In [2]:

```
l3 = ["a", "b", "c"]
l4 = [2, 4, 6]

for i in range(len(l3)):
    print(l3[i])
    print(l4[i])
    print("------")
```

```
a
2
------
b
4
------
c
6
------
```

In [5]:

```
l5 = list(zip(l3,l4))
l5
```

Out[5]:

```
[('a', 2), ('b', 4), ('c', 6)]
```

In [6]:

```
for elem in l5:
    print (elem)
```

```
('a', 2)
('b', 4)
('c', 6)
```

In [8]:

```
for e1, e2 in l5:
    print(e1)
    print(e2)
    print("------")
```

```
a
2
------
b
4
------
c
6
------
```

In [2]:

```
l1 = [["a"], [3,2], [7, "m"]]
l2 = [(5,7), ["x"], ["y"]]

l3 = list(zip(l1, l2))
l3
```

Out[2]:

```
[(['a'], (5, 7)), ([3, 2], ['x']), ([7, 'm'], ['y'])]
```

In [11]:

```
for e1, e2 in l3:
```

```
    print(e1)
```

```
['a']
[3, 2]
[7, 'm']
```

In [15]:

```
for a1, a2 in l3:
    print(a1)
    print(".......")
    for t1, t2 in e1:
        print(t2)
```

```
['a']
.......
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-2b127e70e328> in <module>
      2     print(a1)
      3     print(".......")
----> 4     for t1, t2 in e1:
      5         print(t2)

ValueError: not enough values to unpack (expected 2, got 1)
```

In [3]:

```
#ZIP PARA GENERAR DICCIONARIOS

print(l1)
print(l2)

d3 = dict(zip(l1, l2))
d3
```

```
[['a'], [3, 2], [7, 'm']]
[(5, 7), ['x'], ['y']]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-96f0f1e72ba7> in <module>
      4 print(l2)
      5
----> 6 d3 = dict(zip(l1, l2))
      7 d3

TypeError: unhashable type: 'list'
```

In [6]:

```
d = {}
d["key"] = 4
print(d)
```

```
{'key': 4}
```

In [1]:

```
l1 = ["x", "y"]
l2 = [0, 1]
l3 = (True, False)

l4 = list(zip(l1, l2, l3)) #Mientras tengan el mismo número de elementos te deja hacerlo
l4
```

Out[1]:

```
[('a', 0, True), ('b', 1, False)]
```

In [2]:

```
l1 = ["x", "y"]
l2 = [0, 1]
l3 = (True, False)
l4 = {6, 5000, 9999} #Por el tipo de ordenación interna que tiene el conjunto diccionario coge primero el 5000 y luego el 6. Si fuese una lista aparecería primero el 6 y luego el 5000, según su orden natural de posición
s = "abdf"

l5 = list(zip(l1, l2, l3, l4, s)) #Hasta donde tienen posiciones iguales lo hace
l5
```

Out[2]:

```
[('x', 0, True, 5000, 'a'), ('y', 1, False, 6, 'b')]
```

In [4]:

```
for pos, (e1, e2, e3, e4, e5) in enumerate(l5):
        print(pos)
        print(e3)
        print("------")
```

```
0
True
------
1
False
------
```

In [12]:

```
l = [3]
l1 = [4, 6]
l.extend(l1)
l
```

Out[12]:

```
[3, 4, 6]
```

In [16]:

```
l = [3]
for i in range(21):
    l.append(i)
    print(l)
```

```
[3, 0]
[3, 0, 1]
[3, 0, 1, 2]
[3, 0, 1, 2, 3]
[3, 0, 1, 2, 3, 4]
[3, 0, 1, 2, 3, 4, 5]
[3, 0, 1, 2, 3, 4, 5, 6]
[3, 0, 1, 2, 3, 4, 5, 6, 7]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [21]:

```
l = [3]
```

```
for i in range(21):
    l.append(i)
print(l)
```

```
[3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [18]:

```
l = [i for i in range(21)] #lo que añado a la lista lo tengo que poner a la izq del for
l
```

Out[18]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [22]:

```
p = 0
l = [p for elem in [2, 5, 9]]#Por cada elemento agrega un cero, porque al recorrer la li
sta en la primera iteración agrega p, en la segunda iteración agrega p y en la tercera it
eración agrega p
l
```

Out[22]:

```
[0, 0, 0]
```

In [23]:

```
p = 0
l = [p for elem in [2, 5, 9] if elem > 4] #el elem 2 no es mayor que 4 por tanto no se i
ncluye ese cero y pasa a la siguiente iteración
l
```

Out[23]:

```
[0, 0]
```

In [26]:

```
p = 0
l = [p for elem in [2, 5, 9] if elem > 4 or isinstance(elem, int)] #Es true si es de tip
o int
l
```

Out[26]:

```
[0, 0, 0]
```

In [27]:

```
p = 0
l = [elem/2 for elem in [2, 5., 9, 17] if isinstance(elem, int)] #Recorremos la coleccio
n por cada elemento de la coleccion (en el primer caso 2) dividiré elem entre dos SI es u
n número int.
l
```

Out[27]:

```
[1.0, 4.5, 8.5]
```

In [28]:

```
o = [2, 5., 9, 17, "x"]
l = [(elem/2) + p for elem in o if isinstance(elem, int)]
k = []

for elem in o:
    if isinstance (elem, int):
        k.append((elem/2) + p)

k
```

Out[28]:

```
[1.0, 4.5, 8.5]
```

In [32]:

```
dict1 = {"a":1, "b": 2, "c": 3, "d": 4, "e": 5}
for (k, v) in dict1.items():
    print("Clave:", k)
    print("Valor:", v)

double_dict1 = {(k*2):(v*2) for (k,v) in dict1.items() if v>2}
double_dict1
```

```
Clave: a
Valor: 1
Clave: b
Valor: 2
Clave: c
Valor: 3
Clave: d
Valor: 4
Clave: e
Valor: 5
```

Out[32]:

```
{'cc': 6, 'dd': 8, 'ee': 10}
```

In [33]:

```
def f1(cont):
    return cont + 1

r = f1(cont=2)
r
```

Out[33]:

```
3
```

In [38]:

```
def f1(cont):
    if cont == 0:
        print("CASO BASE")
        print("Valor de cont la última vez:",cont)
        return cont
    else:
        print("Valor de cont:", cont)
        return f1(cont = cont-1)#Recursividad llamarse una función así misma, lo que la c
onvierte en infinita o hasta que algo la haga parar.

r = f1(cont=2)
r
```

```
Valor de cont: 2
Valor de cont: 1
CASO BASE
Valor de cont la última vez: 0
```

Out[38]:

```
0
```

In [41]:

```
def f1(cont):
    if cont == 0:
        return cont
    else:
        return f1(cont = cont-1)

r = f1(cont=2)
```

```
print(r)
```

```
0
```

#

Iteración 1

cont = 2

el if no lo cumple

else: volvemos a llamar a f1

```
    Iteración 2

    cont = 1

    el if no lo cumple

    else: volvemos a llamar a f1
```

Iteración 3

cont = 0

if lo cumple

return cero por lo tanto va a r y r = 0

-

In [44]:

```
#¿Por qué pass?
def get_data():
    data = []
    return data

def clean_data():
    pass #Para pasar esta función porque ya la rellenaremos esta función mas adelante, po
r ejemplo.

def draw_data():
    pass

get_data()
```

Out[44]:

```
[]
```

In [ ]:

```

```