

Clase Número: 10

Estructura de datos tipo tupla

Hemos desarrollado gran cantidad de algoritmos empleando tipos de datos simples como enteros, flotantes, cadenas de caracteres y estructuras de datos tipo lista.

Vamos a ver otra estructura de datos llamada Tupla.

Una tupla permite almacenar una colección de datos no necesariamente del mismo tipo. Los datos de la tupla son inmutables a diferencia de las listas que son mutables.

Una vez inicializada la tupla no podemos agregar, borrar o modificar sus elementos.

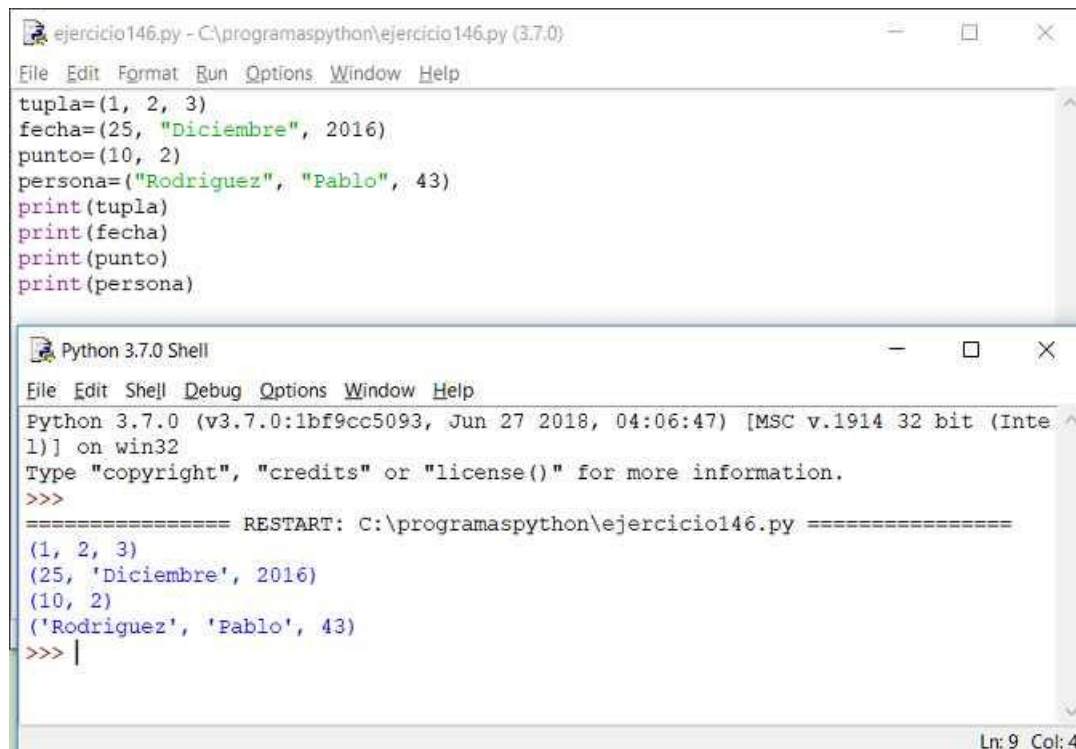
La sintaxis para definir una tupla es indicar entre paréntesis sus valores:

Problema 1:

Definir varias tuplas e imprimir sus elementos.

Programa:

```
tupla=(1, 2, 3)
fecha=(25, "Diciembre", 2016)
punto=(10, 2)
persona=("Rodriguez", "Pablo", 43)
print(tupla)
print(fecha)
print(punto)
print(persona)
```

The image shows a screenshot of a Python IDE. The top window, titled 'ejercicio146.py - C:\programaspython\ejercicio146.py (3.7.0)', contains the following code:

```
tupla=(1, 2, 3)
fecha=(25, "Diciembre", 2016)
punto=(10, 2)
persona=("Rodriguez", "Pablo", 43)
print(tupla)
print(fecha)
print(punto)
print(persona)
```

The bottom window, titled 'Python 3.7.0 Shell', shows the output of the script after execution:

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\programaspython\ejercicio146.py =====
(1, 2, 3)
(25, 'Diciembre', 2016)
(10, 2)
('Rodriguez', 'Pablo', 43)
>>> |
```

Como vemos el lenguaje Python diferencia una tupla de una lista en el momento que la definimos:

```
tupla=(1, 2, 3)
fecha=(25, "Diciembre", 2016)
punto=(10, 2)
persona=("Rodriguez", "Pablo", 43)
```

Utilizamos paréntesis para agrupar los distintos elementos de la tupla.

Podemos acceder a los elementos de una tupla en forma similar a una lista por medio de un subíndice:

```
print(punto[0]) # primer elemento de la tupla
print(punto[1]) # segundo elemento de la tupla
```

Es muy **IMPORTANTE** tener en cuenta que los elementos de la tupla son inmutables, es incorrecto tratar de hacer esta asignación a un elemento de la tupla:

```
punto[0]=70
```

Nos genera el siguiente error:

```
Traceback (most recent call last):
  File "C:/programaspython/ejercicio146.py", line 11, in
    punto[0]=70
TypeError: 'tuple' object does not support item assignment
```

Utilizamos una tupla para agrupar datos que por su naturaleza están relacionados y que no serán modificados durante la ejecución del programa.

Problema 2:

Desarrollar una función que solicite la carga del día, mes y año y almacene dichos datos en una tupla que luego debe retornar. La segunda función a implementar debe recibir una tupla con la fecha y mostrarla por pantalla.

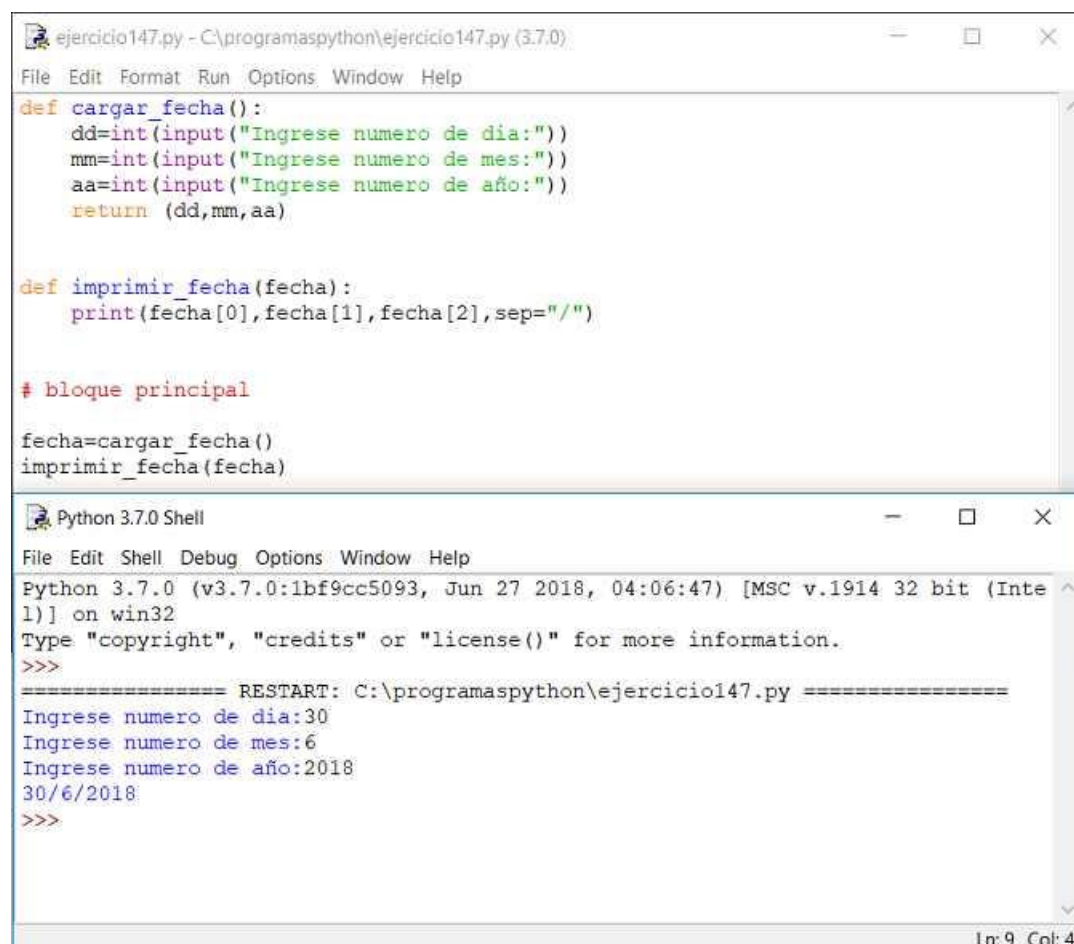
Programa:

```
def cargar_fecha():
    dd=int(input("Ingrese numero de dia:"))
    mm=int(input("Ingrese numero de mes:"))
    aa=int(input("Ingrese numero de año:"))
    return (dd,mm,aa)

def imprimir_fecha(fecha):
    print(fecha[0],fecha[1],fecha[2],sep="/")

# bloque principal

fecha=cargar_fecha()
imprimir_fecha(fecha)
```

The image shows a screenshot of a Python IDE window titled 'ejercicio147.py - C:\programaspython\ejercicio147.py (3.7.0)'. The code editor contains the same Python code as shown in the previous block. Below the code editor is a 'Python 3.7.0 Shell' window. The shell shows the execution of the program, starting with a restart message: 'RESTART: C:\programaspython\ejercicio147.py'. It then prompts for the day, month, and year, with the user entering 30, 6, and 2018 respectively. The final output is '30/6/2018'. The status bar at the bottom right of the shell window indicates 'Ln: 9. Col: 4'.

En la función `cargar_fecha` cargamos tres enteros por teclado y procedemos a crear una tupla indicando entre paréntesis los nombres de las tres variables y procedemos a retornarla:

```
def cargar_fecha():
    dd=int(input("Ingrese numero de dia:"))
```

```
mm=int(input("Ingrese numero de mes:"))
aa=int(input("Ingrese numero de año:"))
return (dd,mm,aa)
```

En el bloque principal llamamos a la función `cargar_fecha` y el valor devuelto se almacena en la variable `fecha` (recordemos que la función devuelve una tupla):

```
fecha=cargar_fecha()
```

Definimos una función que recibe la tupla y procede a mostrar el contenido separado por el caracter `"/"`:

```
def imprimir_fecha(fecha):
    print(fecha[0], fecha[1], fecha[2], sep="/")
```

Conversión de tuplas a listas y viceversa.

Otra herramienta que nos proporciona Python es la conversión de tuplas a listas y viceversa mediante las funciones:

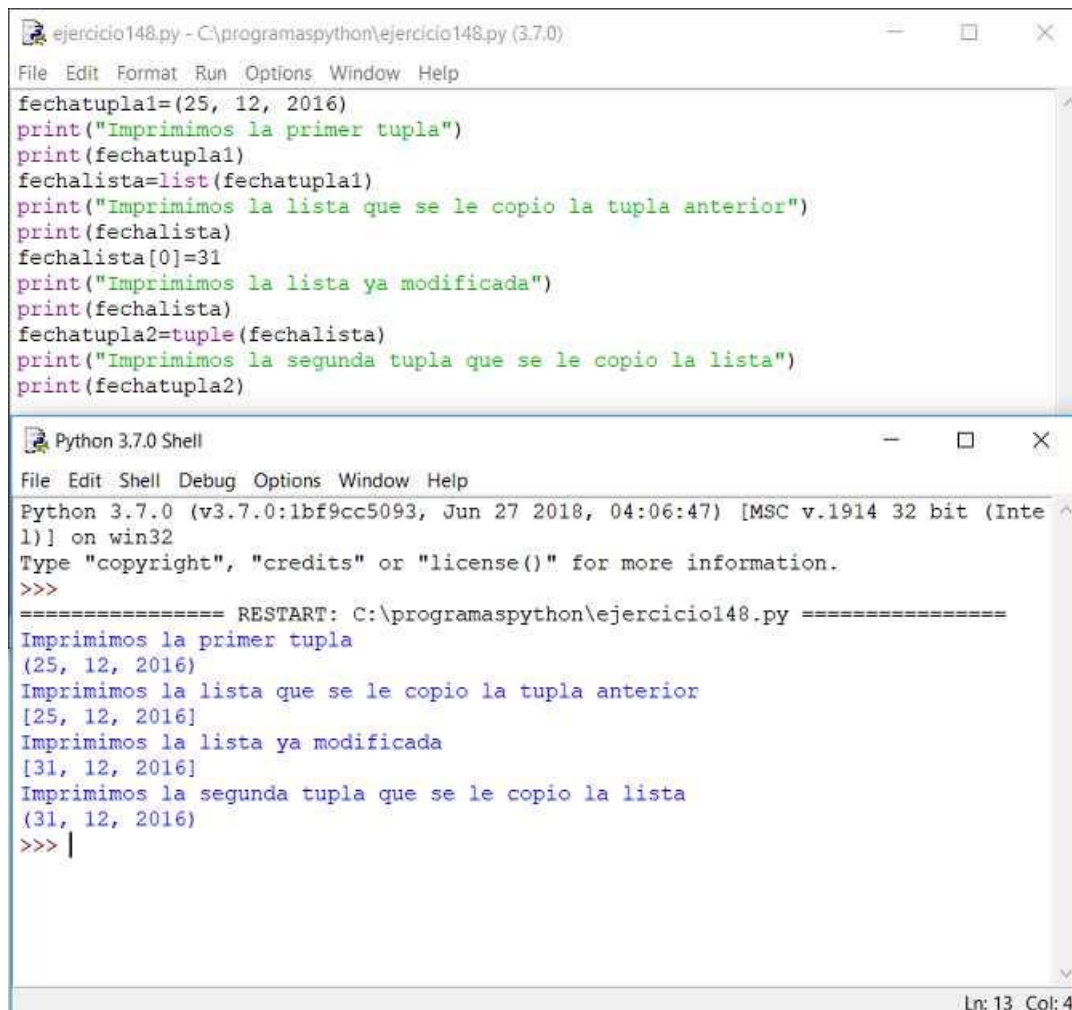
```
list(parametro de tipo tupla)
tuple(parametro de tipo lista)
```

Problema 3:

Definir una tupla con tres valores enteros. Convertir el contenido de la tupla a tipo lista. Modificar la lista y luego convertir la lista en tupla.

Programa:

```
fechatupla1=(25, 12, 2016)
print("Imprimimos la primer tupla")
print(fechatupla1)
fechalista=list(fechatupla1)
print("Imprimimos la lista que se le copio la tupla anterior")
print(fechalista)
fechalista[0]=31
print("Imprimimos la lista ya modificada")
print(fechalista)
fechatupla2=tuple(fechalista)
print("Imprimimos la segunda tupla que se le copio la lista")
print(fechatupla2)
```



```
ejercicio148.py - C:\programaspython\ejercicio148.py (3.7.0)
File Edit Format Run Options Window Help
fechatupla1=(25, 12, 2016)
print("Imprimimos la primer tupla")
print(fechatupla1)
fechalista=list(fechatupla1)
print("Imprimimos la lista que se le copio la tupla anterior")
print(fechalista)
fechalista[0]=31
print("Imprimimos la lista ya modificada")
print(fechalista)
fechatupla2=tuple(fechalista)
print("Imprimimos la segunda tupla que se le copio la lista")
print(fechatupla2)

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\programaspython\ejercicio148.py =====
Imprimimos la primer tupla
(25, 12, 2016)
Imprimimos la lista que se le copio la tupla anterior
[25, 12, 2016]
Imprimimos la lista ya modificada
[31, 12, 2016]
Imprimimos la segunda tupla que se le copio la lista
(31, 12, 2016)
>>> |
```

Empaquetado y desempaquetado de tuplas.

Podemos generar una tupla asignando a una variable un conjunto de variables o valores separados por coma:

```
x=10
y=30
punto=x,y
print(punto)
```

tenemos dos variables enteras x e y. Luego se genera una tupla llamada punto con dos elementos.

```
fecha=(25, "diciembre", 2016)
print(fecha)
dd,mm,aa=fecha
print("Dia",dd)
print("Mes",mm)
print("Año",aa)
```

El desempaquetado de la tupla "fecha" se produce cuando definimos tres variables separadas por coma y le asignamos una tupla:

```
dd,mm,aa=fecha
```

Es importante tener en cuenta de definir el mismo número de variables que la cantidad de elementos de la tupla.

Problema propuesto

1 - Confeccionar un programa con las siguientes funciones:

a) Cargar una lista de 5 enteros.

b) Retornar el mayor y menor valor de la lista mediante una tupla.

Desempaquetar la tupla en el bloque principal y mostrar el mayor y menor.

2 - Confeccionar un programa con las siguientes funciones:

a) Cargar el nombre de un empleado y su sueldo. Retornar una tupla con dichos valores.

b) Una función que reciba como parámetro dos tuplas con los nombres y sueldos de empleados y muestre el nombre del empleado con sueldo mayor. En el bloque principal del programa llamar dos veces a la función de carga y seguidamente llamar a la función que muestra el nombre de empleado con sueldo mayor.

```
# bloque principal
```

```
empleado1=cargar_empleado()
```

```
empleado2=cargar_empleado()
```

```
mayor_sueldo(empleado1, empleado2)
```

Solución a los problemas

problema 1

```
def cargar():
    lista=[]
    for x in range(5):
        valor=int(input("Ingrese valor"))
        lista.append(valor)
    return lista
```

```
def retornar_mayormenor(lista):
    may=lista[0]
    men=lista[0]
    for x in range(1,len(lista)):
        if lista[x]>may:
            may=lista[x]
        else:
            if lista[x]<men:
                men=lista[x]
    return (may,men)
```

bloque principal

```
lista=cargar()
mayor,menor=retornar_mayormenor(lista)
print("Mayor valor de la lista:",mayor)
print("Menor valor de la lista:",menor)
```

problema 2

```
def cargar_empleado():
    nombre=input("Ingrese el nombre del empleado:")
    sueldo=float(input("Ingrese su sueldo:"))
    return (nombre,sueldo)
```

```
def mayor_sueldo(emplead1,empleado2):
    if emplead1[1]>empleado2[1]:
        print(emplead1[0],"tiene mayor sueldo")
    else:
        print(empleado2[0],"tiene mayor sueldo")
```

bloque principal

```
emplead1=cargar_empleado()
empleado2=cargar_empleado()
mayor_sueldo(emplead1,empleado2)
```

Listas y tuplas anidadas

Hemos visto dos estructuras de datos fundamentales en Python que son las listas y las tuplas. La lista es una estructura mutable (es decir podemos modificar sus elementos, agregar y borrar) en cambio una tupla es una secuencia de datos inmutable, es decir una vez definida no puede cambiar.

En Python vimos que podemos definir elementos de una lista que sean de tipo lista, en ese caso decimos que tenemos una lista anidada.

Ahora que vimos tuplas también podemos crear tuplas anidadas.

En general podemos crear y combinar tuplas con elementos de tipo lista y viceversa, es decir listas con componente tipo tupla.

Programa:

```
empleado=["juan", 53, (25, 11, 1999)]
print(empleado)
empleado.append((1, 1, 2016))
print(empleado)
alumno=("pedro",[7, 9])
print(alumno)
alumno[1].append(10)
print(alumno)
```

Por ejemplo definimos la lista llamada empleado con tres elementos: en el primero almacenamos su nombre, en el segundo su edad y en el tercero la fecha de ingreso a trabajar en la empresa (esta se trata de una lista) Podemos más adelante durante la ejecución del programa agregar otro elemento a la lista con por ejemplo la fecha que se fue de la empresa:

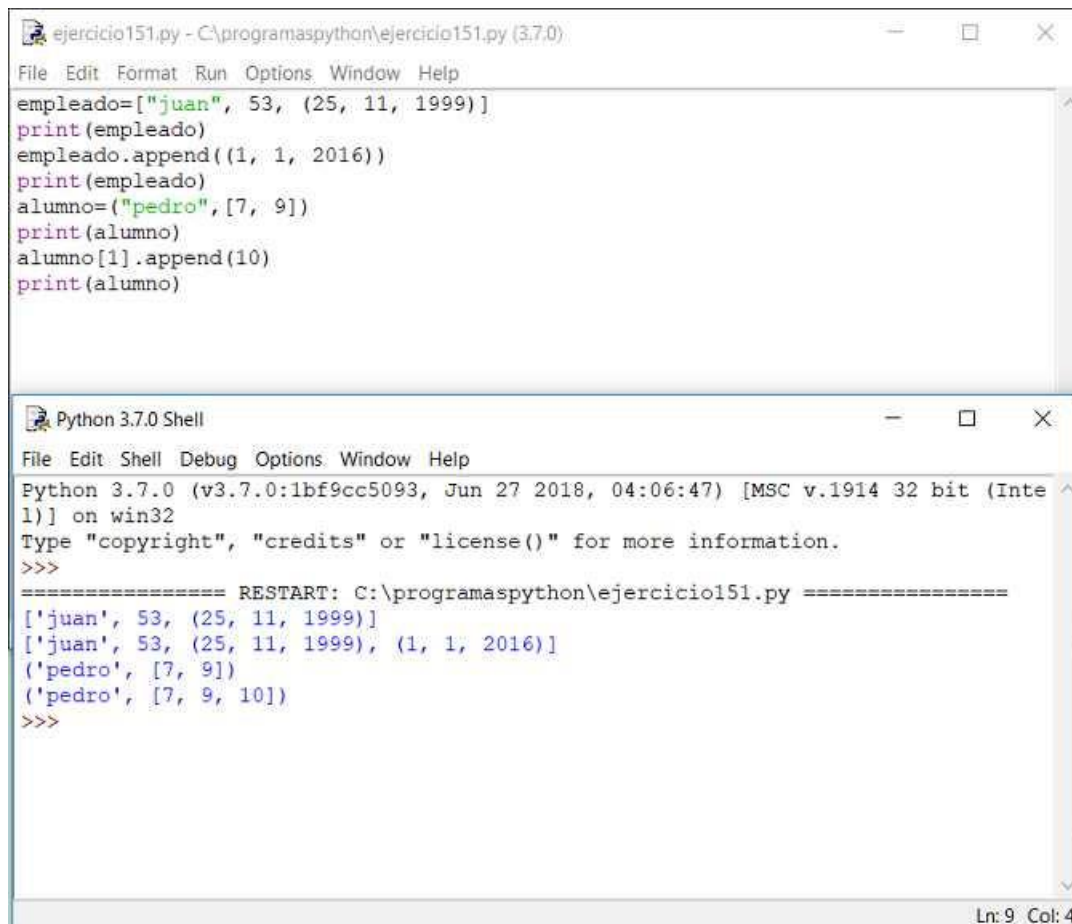
```
empleado=["juan", 53, (25, 11, 1999)]
print(empleado)
empleado.append((1, 1, 2016))
print(empleado)
```

Tenemos definida la tupla llamada alumno con dos elementos, en el primero almacenamos su nombre y en el segundo una lista con las notas que ha obtenido hasta ahora:

```
alumno=("pedro",[7, 9])
print(alumno)
```

Podemos durante la ejecución del programa agregar una nueva nota a dicho alumno:

```
alumno[1].append(10)
print(alumno)
```

The screenshot shows a Python IDE with two windows. The top window, titled 'ejercicio151.py - C:\programaspython\ejercicio151.py (3.7.0)', contains the following code:

```
empleado=["juan", 53, (25, 11, 1999)]
print(empleado)
empleado.append((1, 1, 2016))
print(empleado)
alumno=("pedro", [7, 9])
print(alumno)
alumno[1].append(10)
print(alumno)
```

The bottom window, titled 'Python 3.7.0 Shell', shows the output of the script:

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\programaspython\ejercicio151.py =====
['juan', 53, (25, 11, 1999)]
['juan', 53, (25, 11, 1999), (1, 1, 2016)]
('pedro', [7, 9])
('pedro', [7, 9, 10])
>>>
```

The status bar at the bottom right of the shell window indicates 'Ln: 9 Col: 4'.

Problema 1:

Almacenar en una lista de 5 elementos tuplas que guarden el nombre de un país y la cantidad de habitantes.

Definir tres funciones, en la primera cargar la lista, en la segunda imprimirla y en la tercera mostrar el nombre del país con mayor cantidad de habitantes.

Programa:

```
def cargar_paisespoblacion():
    paises=[]
    for x in range(5):
        nom=input("Ingresar el nombre del pais:")
        cant=int(input("Ingrese la cantidad de habitantes:"))
        paises.append((nom,cant))
    return paises

def imprimir(paises):
    print("Paises y su poblacion")
    for x in range(len(paises)):
        print(paises[x][0],paises[x][1])

def pais_maspoblacion(paises):
    pos=0
    for x in range(1,len(paises)):
        if paises[x][1]>paises[pos][1]:
            pos=x
    print("Pais con mayor cantidad de habitantes:",paises[pos][0])
```

```
# bloque principal
```

```
países=cargar_paisespoblacion()  
imprimir(países)  
país_mas_poblacion(países)
```

En la primer función definimos una lista llamada países y dentro de una estructura repetitiva cargamos un string con el nombre del país y una variable entera con la cantidad de habitantes, luego agregamos un elemento a la lista de tipo tupla:

```
def cargar_paisespoblacion():  
    países=[]  
    for x in range(5):  
        nom=input("Ingresar el nombre del país:")  
        cant=int(input("Ingrese la cantidad de habitantes:"))  
        países.append((nom,cant))  
    return países
```

La segunda función recibe la lista y procedemos a mostrar cada tupla contenida en cada elemento de la lista:

```
def imprimir(países):  
    print("Países y su población")  
    for x in range(len(países)):  
        print(países[x][0],países[x][1])
```

Para identificar el nombre del país con mayor población iniciamos una variable pos con el valor 0 indicando que en dicha posición de la lista se encuentra el país con mayor población, luego dentro del for controlamos las demás tuplas almacenadas en la lista si hay un país con mayor población:

```
def país_mas_poblacion(países):  
    pos=0  
    for x in range(1,len(países)):  
        if países[x][1]>países[pos][1]:  
            pos=x  
    print("País con mayor cantidad de habitantes:",países[pos][0])
```

Problemas propuestos

1 - Almacenar en una lista de 5 elementos los nombres de empleados de una empresa junto a sus últimos tres sueldos (estos tres valores en una tupla)

El programa debe tener las siguientes funciones:

- a) Carga de los nombres de empleados y sus últimos tres sueldos.
- b) Imprimir el monto total cobrado por cada empleado.
- c) Imprimir los nombres de empleados que tuvieron un ingreso trimestral mayor a 10000 en los últimos tres meses.

2 - Se tiene que cargar los votos obtenidos por tres candidatos a una elección.

En una lista cargar en la primer componente el nombre del candidato y en la segunda componente cargar una lista con componentes de tipo tupla con el nombre de la provincia y la cantidad de votos obtenidos en dicha provincia.

Se deben cargar los datos por teclado, pero si se cargaran por asignación tendría una estructura similar a esta:

```
candidatos=[("juan", [("cordoba",100), ("buenos aires",200)]), ("ana",  
[("cordoba",55)]), ("luis", [("buenos aires",20)])]
```

a) Función para cargar todos los candidatos, sus nombres y las provincias con los votos obtenidos.

2) Imprimir el nombre del candidato y la cantidad total de votos obtenidos en todas las provincias.

Solución a los problemas

problema 1

```
def cargar_empleados():
    empleados=[]
    for x in range(5):
        nom=input("Ingrese el nombre del empleado:")
        su1=int(input("Primer sueldo:"))
        su2=int(input("Segundo sueldo:"))
        su3=int(input("Tercer sueldo:"))
        empleados.append([nom, (su1,su2,su3)])
    return empleados

def ganancias(empleados):
    print("Monto total ganado por empleado en los ultimos tres meses")
    for x in range(5):
        total=empleados[x][1][0]+empleados[x][1][1]+empleados[x][1][2]
        print(empleados[x][0],total)

def ganancias_superior10000(empleados):
    print("Empleados con ingresos superiores a 10000 en los ultimos 3 meses")
    for x in range(5):
        total=empleados[x][1][0]+empleados[x][1][1]+empleados[x][1][2]
        if total>10000:
            print(empleados[x][0],total)

# bloque principal

empleados=cargar_empleados()
ganancias(empleados)
ganancias_superior10000(empleados)
```

problema 2

```
def cargar_candidatos():
    candidatos=[]
    for x in range(3):
        nom=input("Ingrese el nombre del candidato:")
        cant=int(input("Los votos de cuantas provincias tiene para cargar?"))
        provincias=[]
        for z in range(cant):
            prov=input("Nombre de provincia:")
            votos=int(input("Cantidad de votos:"))
            provincias.append((prov,votos))
        candidatos.append((nom,provincias))
    return candidatos

def totalvotos_candidato(candidatos):
    for x in range(len(candidatos)):
        suma=0
        for z in range(len(candidatos[x][1])):
            suma=suma+candidatos[x][1][z][1]
        print(candidatos[x][0],suma)

# bloque principal

candidatos=cargar_candidatos()
```

```
totalvotos_candidato(candidatos)
```

Variantes de la estructura repetitiva for para recorrer tuplas y listas

Hasta ahora siempre que recorremos una lista o una tupla utilizando un for procedemos de la siguiente manera:

```
lista=[2, 3, 50, 7, 9]

for x in range(len(lista)):
    print(lista[x])
```

Esta forma de recorrer la lista es utilizada obligatoriamente cuando queremos modificar sus elementos como podría ser:

```
lista=[2, 3, 50, 7, 9]

print(lista) # [2, 3, 50, 7, 9]

for x in range(len(lista)):
    if lista[x]<10:
        lista[x]=0

print(lista) # [0, 0, 50, 0, 0]
```

Ahora veremos una segunda forma de acceder a los elementos de una lista con la estructura repetitiva for sin indicar subíndices.

```
lista=[2, 3, 50, 7, 9]

for elemento in lista:
    print(elemento)
```

Como podemos ver la instrucción for requiere una variable (en este ejemplo llamada elemento), luego la palabra clave in y por último el nombre de la lista. El bloque del for se ejecuta tantas veces como elementos tenga la lista, y en cada vuelta del for la variable elemento almacena un valor de la lista.

Problema 1:

Confeccionar un programa que permita la carga de una lista de 5 enteros por teclado.

Luego en otras funciones:

- 1) Imprimirla en forma completa.
- 2) Obtener y mostrar el mayor.
- 3) Mostrar la suma de todas sus componentes.

Utilizar la nueva sintaxis de for vista en este concepto.

Programa:

```
def cargar():
    lista=[]
    for x in range(5):
        num=int(input("Ingrese un valor:"))
        lista.append(num)
    return lista
```

```

def imprimir(lista):
    print("Lista completa")
    for elemento in lista:
        print(elemento)

def mayor(lista):
    may=lista[0]
    for elemento in lista:
        if elemento>may:
            may=elemento
    print("El elemento mayor de la lista es",may)

def sumar_elementos(lista):
    suma=0
    for elemento in lista:
        suma=suma+elemento
    print("La suma de todos sus elementos es",suma)

# bloque principal

lista=cargar()
imprimir(lista)
mayor(lista)
sumar_elementos(lista)

```

En la carga planteamos un for que se repita cinco veces y es imposible recorrer la lista con el for ya que antes de entrar al for la lista se define vacía:

```

def cargar():
    lista=[]
    for x in range(5):
        num=int(input("Ingrese un valor:"))
        lista.append(num)
    return lista

```

Las funciones imprimir, mayor y sumar_elementos son lugares muy convenientes para acceder a los elementos de la lista con la nueva sintaxis del for:

```

def imprimir(lista):
    print("Lista completa")
    for elemento in lista:
        print(elemento)

def mayor(lista):
    may=lista[0]
    for elemento in lista:
        if elemento>may:
            may=elemento
    print("El elemento mayor de la lista es",may)

def sumar_elementos(lista):
    suma=0
    for elemento in lista:
        suma=suma+elemento
    print("La suma de todos sus elementos es",suma)

```

Problema 2:

Almacenar en una lista de 5 elementos las tuplas con el nombre de empleado y su sueldo.

Implementar las funciones:

- 1) Carga de empleados.
- 2) Impresión de los empleados y sus sueldos.
- 3) Nombre del empleado con sueldo mayor.
- 4) Cantidad de empleados con sueldo menor a 1000.

Programa:

```
def cargar():
    empleados=[]
    for x in range(5):
        nombre=input("Nombre del empleado:")
        sueldo=int(input("Ingrese el sueldo:"))
        empleados.append((nombre,sueldo))
    return empleados

def imprimir(empleados):
    print("Listado de los nombres de empleados y sus sueldos")
    for nombre,sueldo in empleados:
        print(nombre,sueldo)

def mayor_sueldo(empleados):
    empleado=empleados[0]
    for emp in empleados:
        if emp[1]>empleado[1]:
            empleado=emp
    print("Empleado con mayor sueldo:",empleado[0],"su sueldo es",empleado[1])

def sueldos_menor1000(empleados):
    cant=0
    for empleado in empleados:
        if empleado[1]<1000:
            cant=cant+1
    print("Cantidad de empleados con un sueldo menor a 1000 son:",cant)

# bloque principal

empleados=cargar()
imprimir(empleados)
mayor_sueldo(empleados)
sueldos_menor1000(empleados)
```

La carga de la lista con elementos de tipo tupla ya la conocemos de conceptos anteriores:

```
def cargar():
    empleados=[]
    for x in range(5):
        nombre=input("Nombre del empleado:")
        sueldo=int(input("Ingrese el sueldo:"))
        empleados.append((nombre,sueldo))
    return empleados
```

Algo nuevo podemos ver ahora en el for para recuperar cada tupla almacenada en la lista.

Podemos ver que desempaquetamos la tupla que devuelve el for en cada vuelta en las variables

nombre y sueldo. Esto nos facilita la impresión de los datos sin tener que indicar subíndices para los elementos de la tupla:

```
def imprimir(empleados):
    print("Listado de los nombres de empleados y sus sueldos")
    for nombre,sueldo in empleados:
        print(nombre,sueldo)
```

Para obtener el sueldo mayor y el nombre del empleado definimos una variable local llamada empleado que almacene el primer elemento de la lista empleados.

En cada vuelta del for en la variable emp se almacena una tupla de la lista empleados y procedemos a analizar si el sueldo es mayor al que hemos considerado mayor hasta ese momento, en caso afirmativo actualizamos la variable empleado:

```
def mayor_sueldo(empleados):
    empleado=empleados[0]
    for emp in empleados:
        if emp[1]>empleado[1]:
            empleado=emp
    print("Empleado con mayor sueldo:",empleado[0],"su sueldo es", empleado[1])
```

En forma similar procesamos la lista para contar la cantidad de empleados con un sueldo menor a 1000:

```
def sueldos_menor1000(empleados):
    cant=0
    for empleado in empleados:
        if empleado[1]<1000:
            cant=cant+1
    print("Cantidad de empleados con un sueldo menor a 1000 son:",cant)
```

Desde el bloque principal procedemos a llamar a las distintas funciones:

```
# bloque principal

empleados=cargar()
imprimir(empleados)
mayor_sueldo(empleados)
sueldos_menor1000(empleados)
```

Problemas propuestos

1 - Definir una función que cargue una lista con palabras y la retorne.

Luego otra función tiene que mostrar todas las palabras de la lista que tienen más de 5 caracteres.

2 - Almacenar los nombres de 5 productos y sus precios. Utilizar una lista y cada elemento una tupla con el nombre y el precio.

Desarrollar las funciones:

a) Cargar por teclado.

b) Listar los productos y precios.

c) Imprimir los productos con precios comprendidos entre 10 y 15.

Solución a los problemas

problema 1

```
def cargar():
    palabras=[]
    cant=int(input("Cuantas palabras quiere cargar?"))
    for x in range(cant):
        pal=input("Ingrese palabra:")
        palabras.append(pal)
    return palabras

def palabras_mas5(palabras):
    print("Palabras ingresadas con mas de 5 caracteres")
    for palabra in palabras:
        if len(palabra)>5:
            print(palabra)

# bloque principal

palabras=cargar()
palabras_mas5(palabras)
```

problema 2

```
def cargar():
    productos=[]
    for x in range(5):
        nombre=input("Ingrese el nombre del producto:")
        precio=int(input("Ingrese el precio:"))
        productos.append((nombre,precio))
    return productos

def imprimir(productos):
    print("Listado de productos y precios")
    for nombre,precio in productos:
        print(nombre,precio)

def imprimir_entre10y15(productos):
    print("Listado de productos que tienen un precio comprendido entre 10 y 15")
    for nombre,precio in productos:
        if precio>=10 and precio<=15:
            print(nombre,precio)

# bloque principal

productos=cargar()
imprimir(productos)
```