

# 05

UNIDAD

# P1

## Programación 1

### — ESTRUCTURA DE DATOS TIPO TUPLA —



- | Estructura de datos tipo tupla
- | Listas y tuplas anidadas
- | Variantes repetitivas para recorrer tuplas y listas



- | En la clase de hoy aprenderás a diferenciar entre listas y tuplas y su implementación para resolver tus problemas
- | Profundizarás tu comprensión sobre la aplicación de funciones repetitivas para recorrer tuplas y listas

MÓDULO  
DIDÁCTICO  
2023

ISSD

# 10

CLASE

## INTRODUCCIÓN A LA CLASE

El mundo de los algoritmos es apasionante y no vislumbra fronteras. Hace apenas unas décadas era impensado soñar con los avances que la programación y la tecnología alcanzarían al día de hoy.

Sin embargo, para poder realizar tareas más complicadas es necesario que sigas profundizando tu lógica y ejercitando tu pensamiento mediante desafíos y algoritmos cada vez más complejos.

En la clase de hoy seguiremos estudiando la resolución de situaciones problemáticas mediante la aplicación de tuplas y listas.

La clase se estructura en dos grandes temas:

- | En el primero de ellos nos introduciremos a las tuplas y sus diferencias con las listas.

- | En el segundo, trabajaremos con la estructura repetitiva **for** para recorrer listas y tuplas.

Tené en cuenta que la realización de los ejercicios que te vamos proponiendo es fundamental para comprender esta materia.

¡Esperamos que disfrutes la clase de hoy!

Antes de comenzar con la clase, lee el siguiente artículo:

## DESARROLLAN UNA HERRAMIENTA CON INTELIGENCIA ARTIFICIAL PARA LA DETECCIÓN PRECOZ DEL MELANOMA

Un equipo de investigadores de la **Universidad de Málaga (UMA)** ha creado una innovadora herramienta basada en **inteligencia artificial (IA)** y clasificación automática de imágenes para mejorar la precisión en la detección del melanoma, el tipo más agresivo de cáncer de piel. Esta herramienta no solo tiene el potencial de **mejorar la detección temprana del melanoma**, sino que también podría ser utilizada para el diagnóstico de otras enfermedades, como el cáncer de mama.

El equipo de investigadores utilizó un enfoque basado en algoritmos genéticos, que son técnicas computacionales inspiradas en la evolución biológica, junto con la clasificación automática de imágenes. Este método les permitió **analizar y detectar patrones asociados al melanoma** en un conjunto de fotografías.

El objetivo fue comprender el proceso de pensamiento del algoritmo, que guía a la máquina en la toma de decisiones, para que pueda **realizar cálculos y obtener resultados de manera lógica y eficiente**.

No todos los algoritmos son “transparentes” en el sentido de que los usuarios no pueden comprender por qué toman ciertas decisiones o qué pasos han seguido para llegar a un resultado. Algunos algoritmos son considerados “**cajas negras**”. En el caso de esta investigación, los expertos se centraron en explicar los pasos y la lógica del algoritmo **utilizado en la detección de lesiones en la piel, para obtener conclusiones claras y comprensibles**.

Para comprender cómo “razona” el algoritmo, los investigadores desarrollaron un “explicador” que resalta las áreas en las que el sistema se enfoca para detectar el melanoma. Visualmente, se pueden observar los píxeles iluminados en amarillo, lo que revela qué áreas está analizando el algoritmo y qué proceso lógico sigue para determinar si una mancha es un melanoma o no.

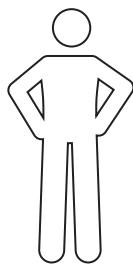
Para probar la efectividad de esta herramienta, los investigadores recopilaron un conjunto de

imágenes que incluían **manchas, lunares, pecas y otras marcas de la piel**, obtenidas de bancos de imágenes gratuitos y bases de datos médicas. Observaron que cuando el algoritmo acertaba, se enfocaba en áreas características de las marcas de la piel, de manera similar a los médicos. Sin embargo, cuando el algoritmo fallaba, se fijaba en áreas menos representativas.

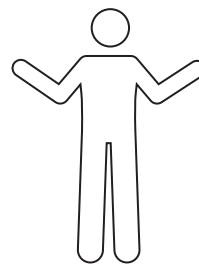
Esta herramienta es capaz de detectar distintas lesiones de la piel, como **melanoma, nevus, queratosis y lunares sanos**. Es importante destacar que este programa **no reemplaza el diagnóstico médico, sino que puede ser una herramienta complementaria**.

*Mundiario.com - julio 2023*

El área de la inteligencia artificial es una de las más sorprendentes, y sin dudas tendrá un desarrollo difícil de pronosticar. Es el camino a seguir si queremos máquinas cada vez más inteligentes.



Algoritmos genéticos, lógica difusa, inteligencia artificial... pero antes es necesario que sigas profundizando tu lógica y ejercitando tu pensamiento mediante desafíos y algoritmos cada vez más complejos, como los que abordaremos durante esta clase.



## 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.

Esto implica que **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**:

*Tupla:* `variable=(1, 2, 3)`

*Lista:* `variable=[1, 2, 3]`

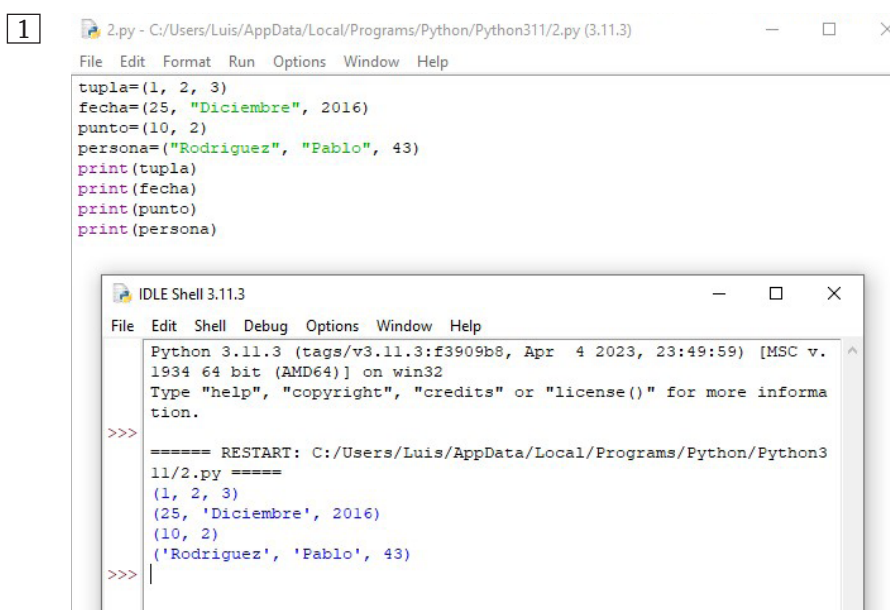
## EJEMPLO 62

### Uso e impresión de tuplas

Definir varias tuplas e imprimir sus elementos.

| Programa: [1]

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



| Una tupla permite almacenar una colección de datos no necesariamente del mismo tipo.

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
```

| Si nos llegáramos a equivocar y, por error asignáramos a un elemento en la tupla; el programa nos mostraría el siguiente error:

Traceback (most recent call last):

File "C:/programasp/python/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.

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

### EJEMPLO 63

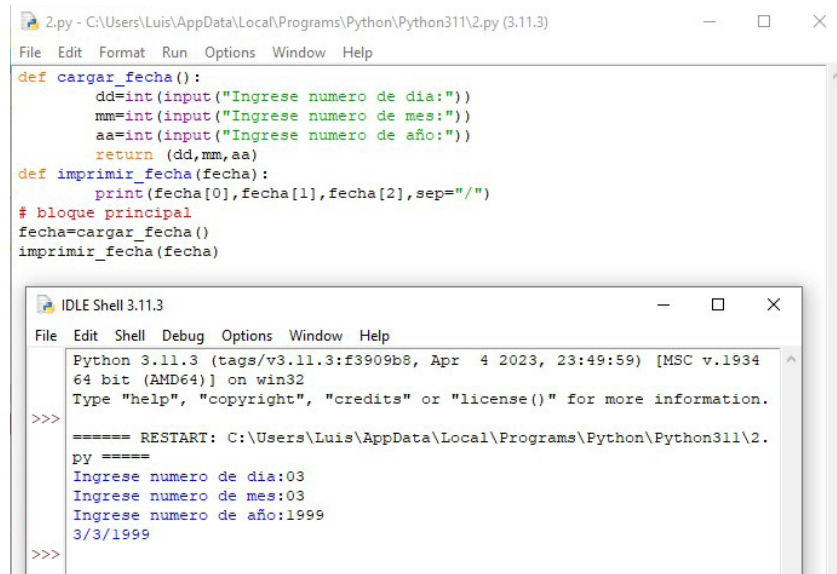
#### Almacén de fechas

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: [2]

```
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)
```

2



The screenshot shows a Python IDE window titled '2.py - C:\Users\Luis\AppData\Local\Programs\Python\Python311\2.py (3.11.3)'. The code in the editor matches the code block above. Below the editor is an 'IDLE Shell 3.11.3' window showing the execution output. The output indicates a restart of the shell and shows the program prompts for day, month, and year, with the user entering 03, 03, and 1999 respectively, resulting in the output '3/3/1999'.

| 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 **carácter "/"**:

```
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)
```

| A continuación veremos una situación que involucre la conversión de tuplas a listas. ¡Vamos!

### EJEMPLO 64

#### Conversión de datos

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: [3]

```
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)
```

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)
```

3

The screenshot shows a Python IDE window titled '2.py - C:\Users\Luis\AppData\Local\Programs\Python\Python311\2.py (3.11.3)'. The script contains the following code:

```

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)

```

Below the script, the 'IDLE Shell 3.11.3' window shows the execution output:

```

Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Luis\AppData\Local\Programs\Python\Python311\2.
py =====
Ingrese numero de dia:03
Ingrese numero de mes:03
Ingrese numero de año:1999
3/3/1999
>>>
== RESTART: C:\Users\Luis\AppData\Local\Programs\Python\Python311\2.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)
>>>

```

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

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

**IMPORTANTE** — Tené en cuenta que deberás definir el mismo número de variables que la cantidad de elementos de la tupla.



### Desempeño 76

Problema en la fiesta: Confeccionar un programa con las siguientes funciones:

- Cargar una lista de 5 enteros.
- 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.



## Desempeño 77

| 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)
```

## 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**: [4]

```
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)
```

4

```
IDLE Shell 3.11.3  
File Edit Shell Debug Options Window Help  
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 6  
4 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:\Users\Luis\AppData\Local\Programs\Python\Python311\2.p  
y =====  
[('juan', 53, (25, 11, 1999))  
 ('juan', 53, (25, 11, 1999), (1, 1, 2016))  
 ('pedro', [7, 9])  
 ('pedro', [7, 9, 10])  
>>> |
```

## EJEMPLO 65

### *Almacén de habitantes por países*

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_mas poblacion(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  
paises=cargar_paisespoblacion()  
imprimir(paises)  
pais_mas poblacion(paises)
```

| 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():
    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
```

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

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

| Para identificar el nombre del país con mayor población iniciamos una variable 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 pais_mas poblacion(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])
```

## 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])
```

| 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]
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]
```

| 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.



### Desempeño 78

| 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| Cargar una lista de 10 elementos enteros.
- 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.



### Desempeño 79

| 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
- b| Imprimir el nombre del candidato y la cantidad total de votos obtenidos en todas las provincias.

## EJEMPLO 66

### Recorridos para listas y tuplas

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)

```

## EJEMPLO 67

### **Empleados y sueldos**

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)
```



### **Desempeño 80**

| 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.



### **Desempeño 81**

| 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.

*Durante esta clase hemos explotado la manipulación de datos  
contenidos en listas y tuplas.*

*La semana que viene nos introduciremos al uso de estructuras de datos  
tipo diccionario para mejorar y optimizar nuestra arquitectura  
de datos en la solución de nuestros ejercicios.*

*¡Te esperamos!*

Si detectás un error del tipo que fuere (falta un punto, un acento, una palabra mal escrita, un error en código, etc.), por favor comunicate con nosotros a [correcciones@issd.edu.ar](mailto:correcciones@issd.edu.ar) e indicanos por cada error que detectes la página y el párrafo.

MUCHAS GRACIAS POR TU APOORTE.

## **SOLUCIONES A LOS DESEMPEÑOS DEL MÓDULO**

*Te recomendamos utilizar esta sección luego de haber intentado por un largo tiempo la resolución y también para verificar tus soluciones.*

### **Solución al desempeño 76**

| Programa:

```
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)
```

### ***Solución al desempeño 77***

| Programa:

```
def cargar_empleado():
    nombre=input("Ingrese el nombre del empleado:")
    sueldo=float(input("Ingrese su sueldo:"))
    return (nombre,sueldo)
def mayor_sueldo(empleador1,empleador2):
    if empleado1[1]>empleado2[1]:
        print(empleado1[0],"tiene mayor sueldo")
    else:
        print(empleado2[0],"tiene mayor sueldo")
# bloque principal
empleado1=cargar_empleado()
empleado2=cargar_empleado()
mayor_sueldo(empleado1,empleado2)
```

### ***Solución al desempeño 78***

| Programa:

```
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)
```



### **Solución al desempeño 79**

| Programa:

```
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)
```

### **Solución al desempeño 80**

| Programa:

```
def cargar():
    palabras=[]
    cant=int(input("Cuántas 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)
```

### ***Solución al desempeño 81***

| Programa:

```
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)
```