



PROGRAMACIÓN
(TDSD214)

ASIGNATURA:	Programación
PROFESOR:	Ing. Lorena Chulde
PERÍODO ACADÉMICO:	2023-B

DEBER

TÍTULO

ARREGLOS

FUNCIONES – PASO DE PARÁMETROS POR TUPLA, LISTAS

Nombres de los estudiantes:

Guerra Lovato Josué

Pérez Orosco Carlos

Soria Ansa Richard

```
>>> a_tuple = ('value1', 'value2', 'value3')
>>> var1, var2, var3 = a_tuple
```

También es una tupla

2023-B

PROPÓSITO DE LA TAREA

Recorrer tuplas mediante estructuras cíclicas para leerlos y visualizar sus valores en pantalla.

TALLER

1. Iterar una tupla

```
Ejercicios Taller > #Ejercicio1_Tuplas > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 tupla = (5, 'Hola curso', 56, 'Juanito', 67)
3 tupla = (5, 'Hola curso', 56, 'Juanito', [1,2,4,6], 67)
4
5 #Iterar la tupla conn for
6 for i in tupla:
7     print(i)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio1_Tuplas"
5
Hola curso
56
Juanito
[1, 2, 4, 6]
67
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

2. Anidar tuplas

```
Ejercicios Taller > #Ejercicio2_Anidar_Tuplas.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 #Anidar tupla
3 tup = 1, 2, ('a', 'b'), 3
4 print(tup)
5 print(tup[2][0], "\n")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio2_Anidar_Tuplas.py"
(1, 2, ('a', 'b'), 3)
a
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

- En el código proporcionado, se define una tupla llamada tup que contiene cuatro elementos: los números enteros 1, 2 y 3, y una tupla anidada que contiene los caracteres 'a' y 'b'. Al imprimir tup, se muestra el contenido completo de la tupla. Luego, al acceder a tup[2][0], se extrae el primer elemento de la tupla anidada, que es 'a', y se imprime. En resumen, el código crea una estructura de datos inmutable que combina diferentes tipos de elementos, y permite acceder a elementos individuales dentro de la tupla anidada mediante indexación.

3. Asignar el valor de una tupla con n elementos a n variables.

Métodos tuplas

count(<obj>)

index(<obj>[,index])

```
Ejercicios Taller > #Ejercicio3_Asignar_Tuplas.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 #Asignar valor de tupla con 3 elementos a 3 variables
3 tupla = ('JUEGO', 'VIDEO', 'COMPUTADOR', 'xd')
4 a, b, c, d = tupla
5 print(a, " ", b, " ", c, " ", d, "\n")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe Ejercicios Taller/#Ejercicio3_Asignar_Tuplas.py
JUEGO VIDEO COMPUTADOR xd

PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

Como tal dentro del Código se asigna un valor a la tupla y como se realiza en clase el valor se que com 3 elementos y 3 variables.

4. Verificar la posición de un elemento:

```
Ejercicios Taller > #Ejercicio4_Tuplas_Verificar.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 # Verificar la posición de elemento
3 print("\nSegunda tupla")
4 tupla2 = (1, 2, 1, 4, 1, 6, 1, 8, 2, 2, 9, 6, 8, 7, 5)
5 count = 0
6 valor = int(input("Ingrese el valor a observar: "))
7 for i in tupla2:
8     if i == valor:
9         count += 1
10 print("El elemento", valor, " se repite:", count, "veces\n")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe Ejercicios Taller/#Ejercicio4_Tuplas_Verificar.py

Segunda tupla
Ingrese el valor a observar: 3
El elemento 3 se repite: 0 veces

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe Ejercicios Taller/#Ejercicio4_Tuplas_Verificar.py

Segunda tupla
Ingrese el valor a observar: 6
El elemento 6 se repite: 2 veces

PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

5. Con un parámetro

```
Ejercicios Taller > #Ejercicio5_Tuplas_Unparametro.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 #Usando la función index con un paramaetro
3 tup1= (7, 7, 7, 3, 5)
4 print(tup1.index(5))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio5_Tuplas_Unparametro.py"
4
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

6. Con dos parámetros

```
Ejercicios Taller > #Ejercicio6_TuplasDosparametros.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 #Usando la función index con dos parametros
3 tup1= (7, 7, 7, 3, 5)
4 print(tup1.index(7,2),"\\n")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio6_TuplasDosparametros.py"
2

PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

7. Imprimir el apellido y el nombre

```
Ejercicios Taller > #Ejercicio7_Nombre_Apellido.py > ...
1 # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2 # Imprimir el nombre y apellido
3 persona =("Josue","Guerra.")
4 nombre, apellido = persona
5 print(apellido)
6 print(nombre)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio7_Nombre_Apellido.py"
Guerra.
Josue
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

8. Imprimir el orden de llegada de los atletas
 - Verificar que es un objeto **enumerate**

```
Ejercicios Taller > #Ejercicio8_Atletas.py > ...
1 # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2 #Metodo enumerate
3 atletas = ("Josue", "Eduard", "Lovato")
4 #posicion = enumerate(atletas)
5 for index, atleta in enumerate(atletas):
6     print("Posición: ",index+1, "atleta: ",atleta)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python311/Python.exe E:\Documentos\Algoritmos\Deberes\Deber 11\Ejercicios Taller/#Ejercicio8_Atletas.py"
Posición: 1 atleta: Josue
Posición: 2 atleta: Eduard
Posición: 3 atleta: Lovato
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

9. Crea una tupla con números, pide un numero por teclado e indica cuantas veces se repite.

```
Ejercicios Taller > #Ejercicio9_Numportecclado.py > ...
1 # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2 #Contando sin la función
3 print("\nTupla")
4 tupla2 = (1, 2, 1, 4, 1, 6, 1, 8, 2, 2, 9, 6, 8, 7, 5)
5 count = 0
6 valor = int(input("Ingrese el valor a observar: "))
7 for i in tupla2:
8     if i == valor:
9         count += 1
10 print("El elemento",valor, " se repite:", count, "veces\n")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python311/Python.exe "e:/Documentos/Algoritmos/Deberes/Deber 11/Ejercicios Taller/#Ejercicio9_Numportecclado.py"

Tupla
Ingrese el valor a observar: 2
El elemento 2 se repite: 3 veces

PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

10. Crea una tupla con números e indica el número con mayor valor y el que menor tenga.

```
Ejercicios Taller > #Ejercicio10_Ordenar_SinFuncion.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  #Crear tupla con números e indicar el número mayor y el menor
3  print("\nSegunda tupla")
4  tupla2 = (2, 5, 100, 7, 2, 6, 8, 35, 28)
5  mayor = tupla2[0]
6  menor = tupla2[0]
7  for num in tupla2:
8      if num > mayor:
9          mayor = num
10     if num < menor:
11         menor = num
12 print("El mayor es:", mayor)
13 print("El menor es:", menor)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/
icios Taller/#Ejercicio10_Ordenar_SinFuncion.py"

Segunda tupla
El mayor es: 100
El menor es: 2
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

11. Usando `max`, `min`

```
Ejercicios Taller > #Ejercicio10_OrdenarMaxMin.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  #Metodo Max y min
3  print("\nPrimer tupla")
4  tupla3 = (2, 5, 100, 7, 2, 6, 8, 35, 28)
5  print("El número mayor es:", max(tupla3))
6  print("El número menor es:", min(tupla3))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio10_OrdenarMaxMin.py"

Primer tupla
El número mayor es: 100
El número menor es: 2
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

La función MAX y MIN realiza el orden de forma automática y eso lo imprime en pantalla, lo cual no es tan necesario que se haga todo el código para poder ordenar valores.

12. Crea una tupla con valores ya predefinidos del 1 al 10, pide un índice por teclado y muestra el valor de la tupla.

```
Ejercicios Taller > #Ejercicio12_Tupla_Valores.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  # Crear la tupla con valores del 1 al 10
3  tupla = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
4  indice = int(input("Ingrese un índice entre 0 y 9: "))
5
6  if 0 <= indice < len(tupla):
7      print("El valor en el índice", indice, "es:", tupla[indice])
8  else:
9      print("Índice fuera de rango.")
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Microsoft/
icios Taller/#Ejercicio12_Tupla_Valores.py"
Ingrese un índice entre 0 y 9: 5
El valor en el índice 5 es: 6
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

13. Convertir una lista en tupla haciendo uso de la función tuple().

```
Ejercicios Taller > #Ejercicio13_Tupla_en_Lista.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  #FUNCIÓN TUPLA -Convertir una lista en tupla
3  valores = [6, 7, 8, 9, 10]
4  print(valores)
5  tupla_lista = tuple(valores)
6  print(tupla_lista)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
icios Taller/#Ejercicio13_Tupla_en_Lista.py"
[6, 7, 8, 9, 10]
(6, 7, 8, 9, 10)
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

Con la función tuple la lista se convierte automáticamente en una tupla, lo que nos facilitaría para la creación de códigos.

14. Crea una lista vacía (pongamos 10 posiciones), pide sus valores y devuelve la suma y la media de los números.

```
Ejercicios Taller > #Ejercicio14_Pormedio_Suma.py > ...
1  #miscelanea-listas
2  #crear una lista vacia, (5) que pida sus valores
3  # y devuelva la suma y el promedio
4  lista = []
5  suma = 0
6  promedio = 0
7  numero = 0
8  for i in range(5):
9      numero = int(input("Ingrese un valor: "))
10     lista.append(numero)
11     suma = suma + numero
12
13     promedio = suma / len(lista)
14     for i in lista:
15         print(i, end=" ")
16     print("\nLa suma de los valores es:", suma)
17     print("El promedio de los valores es:", promedio)
18
19     lista=[5, 7, 3, 6, 2, 8, 9, 11, 4]
20     for i in range(1, len(lista)):
21         aux= lista[i]
22         indice = i
23         while indice >0 and lista[indice-1]> aux:
24             lista[indice] = lista[indice-1]
25             indice = indice - 1
26         lista[indice]=aux
27     print(lista)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/
icios Taller/#Ejercicio14_Pormedio_Suma.py"
Ingrese un valor: 2
Ingrese un valor: 1
Ingrese un valor: 5
Ingrese un valor: 6
Ingrese un valor: 9
2 1 5 6 9
La suma de los valores es: 23
El promedio de los valores es: 4.6
[5, 7, 7, 7, 7, 8, 9, 11, 2]
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

Algoritmos de ordenamiento por inserción

15. Realizar la prueba de escritorio

Forma corta

```
lista = [5,7,1,3,8,4,9,2,6]
for i in range(1, len(lista)):
    actual = lista[i]
    indice = i
    while indice > 0 and lista[indice-1] > actual:
        lista[indice] = lista[indice-1]
        indice = indice-1
    lista[indice] = actual
print(lista)
```

Prueba de escritorio de forma escrita**Iteración 0:**

La lista es [5, 7, 1, 3, 8, 4, 9, 2, 6].

No hay ningún valor actual ni índice en esta etapa.

La lista no cambia en esta etapa.

Iteración 1:

El valor actual es 7.

El índice es 1.

Como 7 es mayor que 5, no hay cambios en la lista.

La lista sigue siendo [5, 7, 1, 3, 8, 4, 9, 2, 6].

Iteración 2:

El valor actual es 1.

El índice es 2.

Comparamos 1 con 7 y 5. 1 es menor que ambos.

Desplazamos 7 y 5 una posición a la derecha.

Actualizamos el índice a 0.

La lista se convierte en [1, 5, 7, 3, 8, 4, 9, 2, 6].

Iteración 3:

El valor actual es 3.

El índice es 3.

Comparamos 3 con 7, 5 y 1. 3 es menor que 7 pero mayor que 1.

Desplazamos 7 y 5 una posición a la derecha.

Colocamos 3 en la posición correcta.

La lista se convierte en [1, 3, 5, 7, 8, 4, 9, 2, 6].

Iteración 4:

El valor actual es 8.

El índice es 4.

Como 8 es mayor que 7, no hay cambios en la lista.

La lista sigue siendo [1, 3, 5, 7, 8, 4, 9, 2, 6].

Iteración 5:

El valor actual es 4.

El índice es 5.

Comparamos 4 con 8, 7, 5 y 3. 4 es menor que todos.

Desplazamos 8, 7 y 5 una posición a la derecha.

Colocamos 4 en la posición correcta.

La lista se convierte en [1, 3, 4, 5, 7, 8, 9, 2, 6].

Iteración 6:

El valor actual es 9.

El índice es 6.

Como 9 es mayor que 8, no hay cambios en la lista.

La lista sigue siendo [1, 3, 4, 5, 7, 8, 9, 2, 6].

Iteración 7:

El valor actual es 2.

El índice es 7.

Comparamos 2 con 9, 8, 7, 5 y 4. 2 es menor que todos.

Desplazamos 9, 8, 7, 5 y 4 una posición a la derecha.

Colocamos 2 en la posición correcta.

La lista se convierte en [1, 2, 3, 4, 5, 7, 8, 9, 6].

Iteración 8:

El valor actual es 6.

El índice es 8.

Comparamos 6 con 9, 8 y 7. 6 es menor que 9 pero mayor que 2.

Desplazamos 9, 8 y 7 una posición a la derecha.

Colocamos 6 en la posición correcta.

La lista se convierte en [1, 2, 3, 4, 5, 6, 7, 8, 9].

- Después de la última iteración, la lista está ordenada correctamente. Por lo tanto, el resultado final es [1, 2, 3, 4, 5, 6, 7, 8, 9].

Prueba de escritorio en formato de tabla:

	A	B	C	D	E	F
1	Iteración	Lista Original	Valor Actual	Índice	Lista Actualizada	
2	0	[5, 7, 1, 3, 8, 4, 9, 2, 6]				
3	1	[5, 7, 1, 3, 8, 4, 9, 2, 6]	7	1	[5, 7, 1, 3, 8, 4, 9, 2, 6]	
4	2	[1, 5, 7, 3, 8, 4, 9, 2, 6]	1	2	[1, 5, 7, 3, 8, 4, 9, 2, 6]	
5	3	[1, 3, 5, 7, 8, 4, 9, 2, 6]	3	3	[1, 3, 5, 7, 8, 4, 9, 2, 6]	
6	4	[1, 3, 5, 7, 8, 4, 9, 2, 6]	8	4	[1, 3, 5, 7, 8, 4, 9, 2, 6]	
7	5	[1, 3, 4, 5, 7, 8, 9, 2, 6]	4	5	[1, 3, 4, 5, 7, 8, 9, 2, 6]	
8	6	[1, 3, 4, 5, 7, 8, 9, 2, 6]	9	6	[1, 3, 4, 5, 7, 8, 9, 2, 6]	
9	7	[1, 2, 3, 4, 5, 7, 8, 9, 6]	2	7	[1, 2, 3, 4, 5, 7, 8, 9, 6]	
10	8	[1, 2, 3, 4, 5, 6, 7, 8, 9]	6	8	[1, 2, 3, 4, 5, 6, 7, 8, 9]	
11						

Ejecución del código:

```

Ejercicios Taller > #Ejercicio16_OrdenamientoInsercion_GuerraJosue.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  lista = [5,7,1,3,8,4,9,2,6]
3
4  for i in range(1, len(lista)):
5      actual = lista[i]
6      indice = i
7      while indice > 0 and lista[indice-1] > actual:
8          lista[indice] = lista[indice-1]
9          indice = indice-1
10     lista[indice] = actual
11     print(lista)
12
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/
icios Taller/#Ejercicio16_OrdenamientoInsercion_GuerraJosue.py"
[1, 2, 3, 4, 5, 6, 7, 8, 9]
PS E:\Documentos\Algoritmos\Deberes\Deber 11>

```

Como se puede observar el código mostrara la lista de forma ordenada acomodando cada valor desde el menor hasta el mayor.

16. Realizar la prueba de escritorio

Función para mostrar estado de la lista

```

def mostrarLista(lista, lon):
    listaordenada=""
    for i in range(0,lon):
        listaordenada+=str(lista[i])+" "
    print(listaordenada)

arreglo = [5,2,4,1,3];
#Recorrer el arreglo
for i in range(1,len(arreglo)):
    clave = arreglo[i]
    j = i-1
    #Comparar el valor seleccionado con todos los valores anteriores
    while (j>=0 and arreglo[j] > clave):
        #Insertar el valor donde corresponda
        arreglo[j+1] = arreglo[j]
        j = j-1
    arreglo[j+1] = clave
    mostrarLista(arreglo, len(arreglo))
mostrarLista(arreglo, len(arreglo))

```

Prueba de escritorio de forma escrita:

Se define la función `mostrarLista(lista, lon)` que toma dos argumentos: `lista`, que es la lista que se mostrará, y `lon`, que es la longitud de la lista.

Se define el arreglo `[5,2,4,1,3]`.

Se ejecuta un bucle `for` que itera sobre los índices del arreglo, comenzando desde 1 hasta la longitud del arreglo (`len(arreglo)`).

En cada iteración, se toma el elemento en la posición `i` y se almacena en la variable `clave`.

Se inicializa `j` con el valor `i-1`.

Se ejecuta un bucle `while` que compara el valor de `arreglo[j]` con `clave` y mueve los elementos mayores hacia la derecha en la lista.

Una vez que se encuentra la posición correcta para insertar `clave`, se inserta en `arreglo[j+1]`.

Se llama a la función `mostrarLista()` para mostrar el estado actual del arreglo en esa iteración.

Después del bucle `for`, se llama a la función `mostrarLista()` una vez más para mostrar el estado final del arreglo.

Primera iteración (i = 1):

arreglo es `[2,5,4,1,3]`.

Segunda iteración (i = 2):

arreglo es `[2,4,5,1,3]`.

Tercera iteración (i = 3):

arreglo es `[1,2,4,5,3]`.

Cuarta iteración (i = 4):

arreglo es `[1,2,3,4,5]`.

Quinta iteración (i = 5):

arreglo sigue siendo `[1,2,3,4,5]`.

- El resultado final es `[1,2,3,4,5]`. Cada línea impresa en el proceso muestra el estado actual del arreglo en cada iteración.

Prueba de escritorio en formato de tabla:

Iteración	Estado de la Lista	clave	j	Comentarios
0	5 2 4 1 3			Estado inicial del arreglo
1	2 5 4 1 3	2	0	Ningún cambio, 2 ya está en su lugar
2	2 4 5 1 3	4	1	4 se mueve una posición a la izquierda
3	1 2 4 5 3	1	0	1 se mueve tres posiciones a la izquierda
4	1 2 3 4 5	3	2	3 se mueve una posición a la izquierda
	1 2 3 4 5			Resultado final

Ejecución del código:

```
Ejercicios Taller > #Ejercicio16_EstadodeLista.py > ...
1  # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2  #Función para mostrar estado de la lista
3
4  def mostrarLista(lista, lon):
5      listaordenada=""
6      for i in range(0,lon):
7          listaordenada+=str(lista[i])+ " "
8      print(listaordenada)
9
10 arreglo = [5,2,4,1,3];
11 #Recorrer el arreglo
12 for i in range(1,len(arreglo)):
13     clave = arreglo[i]
14     j = i-1
15     while (j>=0 and arreglo[j] > clave):|
16         arreglo[j+1] = arreglo[j]
17         j = j-1
18     arreglo[j+1] = clave
19     mostrarLista(arreglo, len(arreglo))
20 mostrarLista(arreglo, len(arreglo))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Mi
Taller/#Ejercicio16_EstadodeLista.py"
2 5 4 1 3
2 4 5 1 3
1 2 4 5 3
1 2 3 4 5
1 2 3 4 5
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

El código implementa el algoritmo de ordenamiento de inserción en Python. Comienza con una lista desordenada `arreglo`, y luego itera sobre cada elemento de la lista, comenzando desde el segundo elemento ($i = 1$). Dentro del bucle `for`, el código selecciona un elemento como la "clave" y lo compara con los elementos anteriores en la lista, desplazando los elementos mayores hacia la derecha para hacer espacio para la inserción de la clave en su posición correcta. Finalmente, se llama a la función `mostrarLista()` para imprimir el estado actualizado de la lista en cada iteración del algoritmo. Una vez que el bucle ha terminado, se llama a `mostrarLista()` nuevamente para imprimir la lista completamente ordenada.

TAREA

1. Crea una tupla con los meses del año, pide números al usuario, si el número está entre 1 y la longitud máxima de la tupla, muestra el contenido de esa posición sino muestra un mensaje de error.
El programa termina cuando el usuario introduce un cero.

```
Ejercicios Tarea > Ejercicio_1_Tarea.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  # Ejercicios tarea
3  # Crea una tupla con los meses del año, pide números
4  # al usuario, si el número está entre 1 y la longitud máxima de la tupla,
5  # muestra el contenido de esa posición sino muestra un mensaje de error.
6  # El programa termina cuando el usuario introduce un cero.
7
8  tuplaMeses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto",
9               "Septiembre", "Octubre", "Noviembre", "Diciembre")
10
11  numeroMes = 1
12
13  while numeroMes != 0:
14      numeroMes = int(input("Ingresa un número del 1 al 12: "))
15      if numeroMes == 0:
16          print("Adios")
17          exit(1)
18      elif 1 <= numeroMes <= len(tuplaMeses):
19          print("El mes correspondiente a ", numeroMes, " es ", tuplaMeses[numeroMes - 1])
20      else:
21          print("Recuerda que es un número entre 1 y 12")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Microsoft/WindowsApps/python3.11.exe '
Tarea/Ejercicio_1_Tarea.py'
Ingresa un número del 1 al 12: 5
El mes correspondiente a 5 es Mayo
Ingresa un número del 1 al 12: 6
El mes correspondiente a 6 es Junio
Ingresa un número del 1 al 12: 8
El mes correspondiente a 8 es Agosto
Ingresa un número del 1 al 12: 0
Adios
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

El código presenta un bucle interactivo que solicita al usuario ingresar un número del 1 al 12, correspondiente a un mes del año. Si el usuario ingresa 0, el programa finaliza con el mensaje "Adios". Si el número ingresado está dentro del rango válido (1 al 12), se imprime el nombre del mes correspondiente utilizando la tupla `tuplaMeses`. Si el número ingresado está fuera de ese rango, se muestra un mensaje recordando al usuario el rango permitido. La ejecución continúa hasta que el usuario ingresa 0 para salir del programa. La tupla `tuplaMeses` contiene los nombres de los meses en orden, y se utiliza para obtener el nombre del mes correspondiente al número ingresado por el usuario. La función `exit(1)` se utiliza para finalizar el programa con un código de salida 1.

2. Implementa una función `siguiente_mayor(lista)`, a la que se le pase como argumento una lista de números enteros y reemplace cada número por el siguiente más grande de la lista.

```
Ejercicios Tarea > Ejercicio_2_Tarea.py > ...
1  # Ejercicios Carlos Perez, JOsue Guerra, Richard Soria
2  # Ejercicios Tarea
3
4  def siguiente_mayor (lista):
5      lista_nueva = []
6      for i in range (len(lista)):
7          siguiente_numero = (lista[i] + 1)
8          lista_nueva.append(siguiente_numero)
9      return lista_nueva
10
11 # Programa Principal
12
13 lista = []
14
15 longitud = int (input ("Ingresa la longitud de la lista: "))
16
17 for i in range (longitud):
18     elemento = int (input ("Ingresa un numero entero: "))
19     lista.append (elemento)
20 print ("La lista es: ", lista)
21 resultado = siguiente_mayor (lista)
22 print ("La lista modificada es: ", resultado)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/
Tarea/Ejercicio_2_Tarea.py"
Ingresa la longitud de la lista: 6
Ingresa un numero entero: 1
Ingresa un numero entero: 5
Ingresa un numero entero: 9
Ingresa un numero entero: 6
Ingresa un numero entero: 3
Ingresa un numero entero: 5
La lista es: [1, 5, 9, 6, 3, 5]
La lista modificada es: [2, 6, 10, 7, 4, 6]
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

El código define una función llamada `siguiente_mayor()` que toma una lista de números como entrada y devuelve una nueva lista donde cada elemento es el número siguiente al correspondiente en la lista original. Luego, en el programa principal, se solicita al usuario ingresar la longitud de la lista y los elementos de la lista. Posteriormente, se llama a la función `siguiente_mayor()` pasando la lista ingresada como argumento, y se imprime la lista original y la lista modificada. En resumen, el programa toma una lista de números y devuelve una lista donde cada número ha sido incrementado en 1.

3. Escribir una función `poker` que reciba cinco cartas de la baraja francesa e informe (devuelva el valor lógico correspondiente) si esas cartas forman o no un *poker* (es decir que hay 4 cartas con el mismo número).

```
Ejercicios Tarea > Ejercicio_3_Tarea.py > ...
1  # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2  # Ejercicios Tarea
3
4  def poker (cartas):
5      contador = [0] * 14
6      for carta in cartas:
7          contador[carta] += 1
8
9      for frecuencia in contador:
10         if frecuencia == 4:
11             return True
12     return False
13
14 # Programa Principal
15 Cartas = []
16
17 for i in range (5):
18     carta = int (input ("Ingresa el numero de tu carta: "))
19     Cartas.append (carta)
20
21 resultado = poker (Cartas)
22 if resultado:
23     print ("Tienes un poker en tu mano")
24 else:
25     print ("No es un poker")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python39-64/Python.exe E:\Documentos\Algoritmos\Deberes\Deber 11> Tarea/Ejercicio_3_Tarea.py
Ingresa el numero de tu carta: 6
Ingresa el numero de tu carta: 5
Ingresa el numero de tu carta: 3
Ingresa el numero de tu carta: 1
Ingresa el numero de tu carta: 2
No es un poker
PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```


4. El tiempo como tuplas. Proponer una representación con tuplas para representar el tiempo. Escribir una función `sumaTiempo` que reciba dos tiempos dados y devuelva su suma.

```
Ejercicios Tarea > Ejercicio_4_Tarea.py > ...
1  # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2  # Ejercicios Tarea
3
4  def sumaTiempo (tiempo_1, tiempo_2):
5      horas = tiempo_1[0] + tiempo_2[0]
6      minutos = tiempo_1[1] + tiempo_2[1]
7      segundos = tiempo_1[2] + tiempo_2[2]
8
9      minutos += segundos // 60
10     segundos %= 60
11     horas += minutos // 60
12     minutos %= 60
13     return horas, minutos, segundos
14
15     tuplaTiempo = (12, 30, 45)
16     tuplaTiempo_2 = (18, 31, 50)
17
18     resultado = sumaTiempo (tuplaTiempo, tuplaTiempo_2)
19     print ("Tiempo 1: ", tuplaTiempo)
20     print ("Tiempo2: ", tuplaTiempo_2)
21     print ("Suma de los timepos: ", resultado)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/
Tarea/Ejercicio_4_Tarea.py"
Tiempo 1: (12, 30, 45)
Tiempo2: (18, 31, 50)
Suma de los timepos: (31, 2, 35)
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

El código define una función llamada `sumaTiempo` que toma dos tuplas como argumentos, cada una representando una cantidad de tiempo en horas, minutos y segundos. La función suma los tiempos proporcionados y devuelve una nueva tupla que representa la suma de ambos tiempos. Luego, en el programa principal, se definen dos tuplas `tuplaTiempo` y `tuplaTiempo_2`, que representan dos periodos de tiempo. Se llama a la función `sumaTiempo` con estas dos tuplas como argumentos, y se imprime el resultado junto con las dos tuplas originales. En resumen, el programa calcula la suma de dos periodos de tiempo representados por tuplas y muestra el resultado junto con los tiempos originales.

5. Escribir una función `diaSiguienteE` que dada una fecha expresada como la terna (Día, Mes, Año) (donde Día, Mes y Año son números enteros) calcule el día siguiente al dado, en el mismo formato.

Ejercicios Tarea > Ejercicio_5_Tarea.py > ...

```
1  # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2  # Ejercicios Tarea
3
4  def diaSiguiente (dia, mes, año):
5      bisiestro = (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0)
6      dias_por_mes = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
7      if bisiestro:
8          dias_por_mes[2] = 29
9      if 1 <= mes <= 12 and 1 <= dia <= dias_por_mes[mes]:
10         dia_siguiente = dia + 1
11         if dia_siguiente > dias_por_mes[mes]:
12             dia_siguiente = 1
13             mes += 1
14             if mes > 12:
15                 mes = 1
16                 año += 1
17         return dia_siguiente, mes, año
18     else:
19         return None
20
21 fecha_actual = (7, 2, 2024)
22 siguiente_fecha = diaSiguiente (*fecha_actual)
23 if siguiente_fecha:
24     print("Fecha actual:", fecha_actual)
25     print("Día siguiente:", siguiente_fecha)
26 else:
27     print("Fecha no válida.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Microsoft/WindowsApps/PythonSoftwareFoundation/Python311/python.exe C:/Users/josug/AppData/Local/Microsoft/WindowsApps/PythonSoftwareFoundation/Python311/Scripts/python.exe E:\Documentos\Algoritmos\Deberes\Deber 11\Ejercicio_5_Tarea.py
Fecha actual: (7, 2, 2024)
Día siguiente: (8, 2, 2024)
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

El código implementa la función `diaSiguiente`, que calcula la fecha del día siguiente a partir de una fecha dada, considerando años bisiestos. Si la fecha proporcionada es válida, se imprime la fecha actual y la siguiente fecha. De lo contrario, se imprime un mensaje de "Fecha no válida". La función ajusta correctamente los días, meses y años cuando se alcanzan los límites de un mes o un año. En resumen, el programa realiza un cálculo preciso de la fecha del día siguiente y maneja adecuadamente los casos de fechas inválidas.

6. Escribir una función `diaSiguienteT` que dada una fecha expresada como la terna (Día, Mes, Año) (donde Día y Año son números enteros, y Mes es el texto Ene, Feb, ..., Dic, según corresponda) calcule el día siguiente al dado, en el mismo formato.

```
Ejercicios Tarea > Ejercicio_6_Tarea.py > diaSiguienteT
1 # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2 # Ejercicios Tarea
3 def diaSiguienteT (dia, mes, año):
4
5     dias_por_mes = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
6
7     if (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0):
8         dias_por_mes[2] = 29
9
10    if mes < 1 or mes > 12 or dia < 1 or dia > dias_por_mes[mes]:
11        return "FECHA INCORRECTA"
12
13
14    tuplaMeses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre")
15    dia_siguiente = dia + 1
16    if dia_siguiente > dias_por_mes[mes]:
17        dia_siguiente = 1
18        mes += 1
19        if mes > 12:
20            mes = 1
21            año += 1
22
23    fecha_siguiente = (dia_siguiente, tuplaMeses[mes], año)
24    return fecha_siguiente
25 fecha_actual = (7, 2, 2024)
26 fecha_siguiente = diaSiguienteT (*fecha_actual)
27 print(f"Fecha actual: {fecha_siguiente}")
```

Ejecución del código:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Programs/Python/Python310/python.exe Ejercicio_6_Tarea.py
Fecha actual: (8, 'Marzo', 2024)
PS E:\Documentos\Algoritmos\Deberes\Deber 11>
```

El código define la función `diaSiguienteT` para calcular la fecha del día siguiente a partir de una fecha dada, considerando años bisiestos. Verifica la validez de la fecha y ajusta correctamente los días y meses si es necesario. Retorna "FECHA INCORRECTA" si la fecha dada no es válida. En el programa principal, se proporciona una fecha actual y se imprime la fecha del día siguiente obtenida con la función. Este código ofrece una solución precisa y eficiente para calcular la fecha siguiente, teniendo en cuenta diversas condiciones y verificando la validez de la fecha proporcionada.

El siguiente ejercicio se encuentra en la siguiente página.

7. **“Rapid”** es una empresa que entrega artículos de bazar como regalos, peluches, chocolates, perfumes, flores, cartas, etc. Su tienda permite al cliente armar un paquete y enviarlo a la dirección del destinatario,



DETALLE	COSTO
Flores + peluche	\$15,50
Flores + carta	\$7,50
Flores + Chocolates	\$12,25
Flores + perfume	\$22,75

Usted es contratado para automatizar la tienda de RAPID.

- Deberá gestionar los pedidos (CRUD) de los clientes presentándoles el siguiente menú:

```

*****RAPID*****
Qué acción desea realizar?
1.  Registrar compra
2.  Mostrar compra
3.  Mostrar el detalle de la compra
4.  Salir del sistema

Ingrese la opción:
  
```

Si ingresa la **opción 4** debe salir del sistema.

- Si el administrador ingresa en la **opción 1** el sistema debe permitir:
 - Registrar nuevas compras.
 - Si el administrador nuevamente ingresa a la opción 1, se deben registrar nueva compra a los ya existentes.

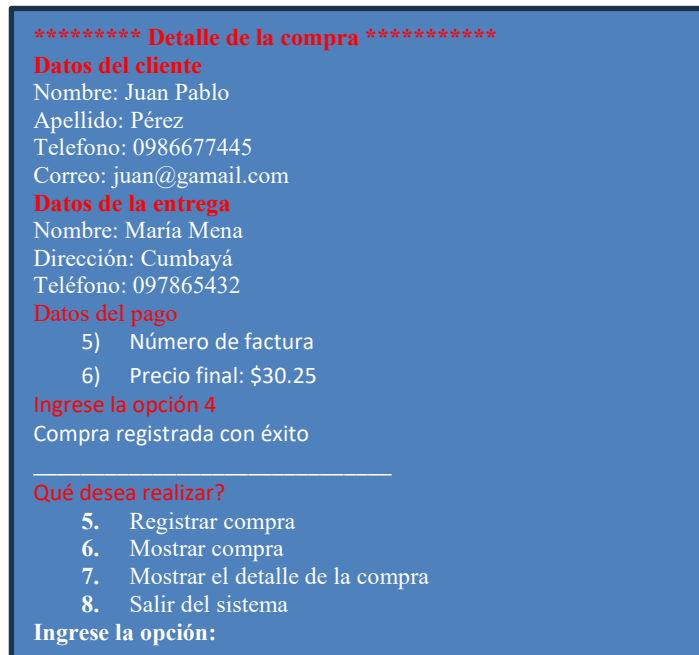
```

***** RAPID *****
***** Nueva compra *****
Ingrese los datos del cliente
Nombre:
Apellido:
Telefono
Correo:
Ingrese los datos de la persona a quién desea enviar el regalo:
Nombre:
Dirección:
Teléfono:
Seleccione el paquete:
1) Flores + peluche costo $15,50
2) Flores + carta costo $7,50
3) Flores + Chocolates costo $12,25
4) Flores + perfume costo $22,75
Ingrese la opción 4
Compra registrada con éxito

Qué desea realizar?
1.  Registrar compra
2.  Mostrar compra
3.  Mostrar el detalle de la compra
4.  Salir del sistema
Ingrese la opción:
  
```

3. Si selecciona la **opción 2** debe permitir:

1. Visualizar todas las compras que se han registrado en la opción 1.



***** Detalle de la compra *****

Datos del cliente
Nombre: Juan Pablo
Apellido: Pérez
Telefono: 0986677445
Correo: juan@gamail.com

Datos de la entrega
Nombre: María Mena
Dirección: Cumbayá
Teléfono: 097865432

Datos del pago
5) Número de factura
6) Precio final: \$30.25

Ingrese la opción 4
Compra registrada con éxito

Qué desea realizar?

- 5. Registrar compra
- 6. Mostrar compra
- 7. Mostrar el detalle de la compra
- 8. Salir del sistema

Ingrese la opción:

2. Y en el caso de que no existan compras registradas, se debe mostrar un mensaje que mencione que no existen compras registradas.

4. Si selecciona la **opción 3** debe visualizar el detalle de la compra con el número de factura, caso contrario mostrar un mensaje de error.

El código y su ejecución se encuentran en la siguiente página.

Ejercicios Tarea > Ejercicio_7_Tarea.py > ...

```

1  # Ejercicios Carlos Perez, Josue Guerra, Richard Soria
2  cantidad_compra = 0
3  factura = 1
4  base_compras = []
5
6
7  def registrarCompra(base_compras, cantidad_compra):
8      global factura
9      print("*****NUEVA COMPRA*****")
10     cantidad_compra += 1
11     print ("Ingrese los datos del cliente:")
12     for i in range(cantidad_compra):
13         detallesCompras = []
14         detallesCompras.insert(0,factura)
15         factura += 1
16         nombre_cliente = str(input("Ingrese el nombre: "))
17         detallesCompras.insert(1, nombre_cliente)
18         apellido_cliente = str(input("Ingrese el apellido: "))
19         detallesCompras.insert(2, apellido_cliente)
20         telefono_cliente = int(input("Ingrese el teléfono: "))
21         detallesCompras.insert(3, telefono_cliente)
22         correo_cliente = str(input("Ingrese el correo: "))
23         detallesCompras.insert(4, correo_cliente)
24         print ("Ingrese los datos de la persona a quién desea enviar el regalo:")
25         nombre_envio = str(input("Ingrese el nombre de envío: "))
26         detallesCompras.insert(5, nombre_envio)
27         apellido_envio = str(input("Ingrese el apellido de envío: "))
28         detallesCompras.insert(6, apellido_envio)
29         telefono_envio = int(input("Ingrese el teléfono de envío: "))
30         detallesCompras.insert(7, telefono_envio)
31         while (True):
32             print ("Seleccione el paquete:")
33             print ("1) Flores + peluche costo $15.50")
34             print ("2) Flores + carta costo $7.50")
35             print ("3) Flores + chocolates costo $12.25")
36             print ("4) Flores + perfume costo $22.75")
37             paquete = int(input("Seleccione una opción: "))
38
39             if (paquete==1):
40                 paquete = "Flores + peluche"
41                 detallesCompras.insert(8, paquete)
42                 precio = 15.50
43                 detallesCompras.insert(9, precio)
44                 break
45             elif (paquete==2):
46                 paquete = "Flores + carta"
47                 detallesCompras.insert(8, paquete)
48                 precio = 7.50
49                 detallesCompras.insert(9, precio)
50                 break
51             elif (paquete==3):
52                 paquete = "Flores + chocolates"
53                 detallesCompras.insert(8, paquete)
54                 precio = 12.25
55                 detallesCompras.insert(9, precio)
56                 break
57             elif (paquete==4):
58                 paquete = "Flores + perfume"
59                 detallesCompras.insert(8, paquete)
60                 precio = 22.75
61                 detallesCompras.insert(9, precio)
62                 break
63             else:
64                 print ("Ingrese un paquete de los disponibles")
65         print("Compra registrada con éxito.")
66         base_compras.append(detallesCompras)
67
68     def mostrarCompra(base_compras):
69         if len(base_compras) > 0:
70             print("*****COMPRAS REGISTRADAS*****")
71             for indice, compra in enumerate(base_compras, start=1):
72                 print("Compra Registrada 0 0 0", indice)
73                 print("Número de factura 0 0 0 0 0", compra[0])
74                 print("Datos del cliente:")
75                 print("Nombre:", compra[1])
76                 print("Apellido:", compra[2])
77                 print("Teléfono:", compra[3])
78                 print("Correo:", compra[4])
79                 print("Datos de la entrega:")
80                 print("Nombre:", compra[5])

```

```

80     print("Nombre:", compra[5])
81     print("Apellido:", compra[6])
82     print("Teléfono:", compra[7])
83     print("Paquete seleccionado:", compra[8])
84     print("Precio:", compra[9])
85     print("Compra registrada con éxito.")
86     print()
87     else:
88         print("No se ha registrado ninguna compra todavía, intente realizar una compra.")
89
90 def mostrarDetallesCompra(base_compras):
91     if len(base_compras) > 0:
92         buscarFactura = int(input("Ingrese la factura a buscar, solo debe indicar las cifras significativas: "))
93         for compra in base_compras:
94             if (compra[0] == buscarFactura):
95                 print("*****DETALLES DE LA COMPRA*****")
96                 print("Compra Registrada 0 0 0", buscarFactura)
97                 print("Factura 0 0 0 0 0", buscarFactura)
98                 print("Datos del cliente:")
99                 print("Nombre:", compra[1])
100                print("Apellido:", compra[2])
101                print("Teléfono:", compra[3])
102                print("Correo:", compra[4])
103                print("Datos de la entrega:")
104                print("Nombre:", compra[5])
105                print("Apellido:", compra[6])
106                print("Teléfono:", compra[7])
107                print("Paquete seleccionado:", compra[8])
108                print("Precio final a pagar:", compra[9])
109                print("Compra registrada con éxito.")
110                print()
111                break
112            else:
113                print("Factura no encontrada. Intente de nuevo.")
114        else:
115            print("No se ha registrado ninguna compra todavía, intente realizar una compra.")
116
117 print ("*****RAPID*****")
118 print ("Bienvenido a Rapid")
119 print ("¿Qué opción desea realizar?")
120 print ("1. Registrar compra")
121 print ("2. Mostrar compra")
122 print ("3. Mostrar el detalle de la compra")
123 print ("4. Salir")
124
125 while (True):
126     opcion = int(input("Ingrese la opción a realizar: "))
127     match opcion:
128         case 1:
129             registrarCompra(base_compras, cantidad_compra)
130         case 2:
131             mostrarCompra(base_compras)
132         case 3:
133             mostrarDetallesCompra(base_compras)
134         case 4:
135             print("Regrese pronto")
136             break
137
138
139
140

```

El código simula un sistema de registro de compras en una tienda llamada "Rapid". Ofrece opciones para registrar nuevas compras, mostrar todas las compras registradas y ver los detalles de una compra específica. Las funciones registrarCompra, mostrarCompra, y mostrarDetallesCompra manejan las diferentes operaciones de registro y visualización de compras. Un bucle principal permite al usuario seleccionar entre estas opciones, con una opción para salir del programa. Este código proporciona una estructura completa para gestionar el proceso de compras en la tienda de manera eficiente y organizada.

Ejecución del código:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Microsoft/Windows/
o_7_Tarea.py"
*****RAPID****
Bienvenido a Rapid
¿Qué opción desea realizar?
1. Registrar compra
2. Mostrar compra
3. Mostrar el detalle de la compra
4. Salir
Ingrese la opción a realizar: 1
*****NUEVA COMPRA****
Ingrese los datos del cliente:
Ingrese el nombre: Josué
Ingrese el apellido: Guerra
Ingrese el teléfono: 0964030442
Ingrese el correo: guerralovatojosue@hotmail.com
Ingrese los datos de la persona a quién desea enviar el regalo:
Ingrese el nombre de envío: compras
Ingrese el apellido de envío: Guerra
Ingrese el teléfono de envío: 0964030442
Seleccione el paquete:
1) Flores + peluche costo $15.50
2) Flores + carta costo $7.50
3) Flores + chocolates costo $12.25
4) Flores + perfume costo $22.75
Seleccione una opción: 1
Compra registrada con éxito.
Ingrese la opción a realizar: 2
*****COMPRAS REGISTRADAS****
Compra Registrada 0 0 0 1
Número de factura 0 0 0 0 0 1
Datos del cliente:
Nombre: Josué
Apellido: Guerra
Teléfono: 964030442
Correo: guerralovatojosue@hotmail.com
Datos de la entrega:
Nombre: compras
Apellido: Guerra
Teléfono: 964030442
Paquete seleccionado: Flores + peluche
Precio: 15.5
Compra registrada con éxito.

Ingrese la opción a realizar: 3
Ingrese la factura a buscar, solo debe indicar las cifras significativas: 1
*****DETALLES DE LA COMPRA****
Compra Registrada 0 0 0 1
Factura 0 0 0 0 0 1
Datos del cliente:
Nombre: Josué
Apellido: Guerra
Teléfono: 964030442
Correo: guerralovatojosue@hotmail.com
Datos de la entrega:
Nombre: compras
Apellido: Guerra
Teléfono: 964030442
Paquete seleccionado: Flores + peluche
Precio final a pagar: 15.5
Compra registrada con éxito.

Ingrese la opción a realizar: 2
*****COMPRAS REGISTRADAS****
Compra Registrada 0 0 0 1
Número de factura 0 0 0 0 0 1
Datos del cliente:
Nombre: Josué
Apellido: Guerra
Teléfono: 964030442
Correo: guerralovatojosue@hotmail.com
Datos de la entrega:
Nombre: compras
Apellido: Guerra
Teléfono: 964030442
Paquete seleccionado: Flores + peluche
Precio: 15.5
Compra registrada con éxito.

Ingrese la opción a realizar: 4
Regrese pronto
PS E:\Documentos\Algoritmos\Deberes\Deber 11> 

```


8. Consultar sobre archivos

¿Qué es el manejo de archivos en Python?

Para empezar con lo básico, se debe entender qué es un archivo. Un archivo es la información o los datos que se recopilan en los distintos dispositivos de almacenamiento de una computadora. Estos pueden ser de música, video, texto, entre otros.

Cuando se trabaja con Python, en general, se dividen los archivos en dos categorías: texto y binarios. Los archivos de texto son texto simple, mientras que los archivos binarios contienen datos que solo pueden ser interpretados por una computadora. Python ofrece formas simples de manipular ambos tipos de archivos. Esto es lo que se conoce como el manejo de archivos en Python.

Python permite y facilita el manejo de archivos, es decir, proporciona a los programadores la capacidad de leer y escribir en estas piezas de datos. Sin embargo, las opciones disponibles para el manejo de archivos con Python no se limitan a esto, ya que existen otras formas de operar con ellos.

El manejo de archivos es un concepto ampliamente utilizado en programación, pero su implementación puede ser complicada. Sin embargo, Python, al igual que con muchas de sus otras funcionalidades, ha logrado simplificar y abreviar este proceso. Esto se debe, entre otras cosas, a cómo Python maneja los archivos binarios. A continuación, se explorará cómo se manejan los archivos en Python.

¿Cómo se manejan los archivos en Python?

Como decíamos, una de las grandes ventajas que ofrece Python es su capacidad para el manejo de archivos, lectura, escritura y otras opciones que veremos a continuación. Primero vamos a empezar por ver como hace que la gestión de archivos sea mucho más sencilla que otros lenguajes de programación.

En Python, cada línea de código incluye una secuencia de caracteres que juntos forman un archivo de texto. Cada una de las líneas de esos archivos termina con un carácter especial denominado EOL o caracteres de final de línea o nueva línea. Con ello, sabemos dónde finaliza y donde comienza cada parte durante la lectura y la escritura de archivos en Python.

Lectura en el manejo de archivos en Python

Python ofrece una función con la que se puede crear un objeto de archivo con varios modos, uno de ellos es el modo de lectura.

El modo de lectura en Python permite devolver el contenido del archivo. De esta manera, el contenido del archivo se mostrará una vez que se ejecute el código relacionado con el modo de lectura.

La función de lectura lee todo el archivo a la vez, pero esto se puede cambiar. Se puede especificar un tamaño de byte con los parámetros de implementación del modo lectura.

Por último, es importante destacar que Python incorpora funciones integradas para leer un archivo línea a línea. Esto simplifica el proceso y evita que los programadores tengan que implementar el mecanismo de lectura de archivos.

Escritura de archivos en Python

El modo o la función de escritura en Python puede habilitarse al mismo tiempo que se ejecuta la función de lectura, o se puede optar por ejecutar ambas de manera independiente, según las necesidades de cada proyecto y programador.

Con la función de escritura, se define el contenido del archivo de texto utilizando la sintaxis de cadenas de Python.

En el caso de que se necesite agregar nuevo contenido a un archivo ya programado mediante la función de escritura, Python permite añadir nuevos contenidos a archivos de texto en el manejo de archivos.

En este caso, aplicar las funciones de lectura y escritura a todo el contenido de nuevo es una opción que consume muchos recursos. Por eso, existe la función de adición, en la cual se pueden añadir los nuevos contenidos para la escritura del archivo [1].

Ejemplo de uso de archivos en Python:

Ejercicios Tarea > Ejercicio_8_EjemploConsulta.py > ...

```
1  # Crear un archivo y escribir en él
2  def escribir_archivo(nombre_archivo, contenido):
3      with open(nombre_archivo, 'w') as archivo:
4          archivo.write(contenido)
5          print(f'Se ha escrito el contenido en el archivo {nombre_archivo}')
6
7  # Leer un archivo
8  def leer_archivo(nombre_archivo):
9      try:
10         with open(nombre_archivo, 'r') as archivo:
11             contenido = archivo.read()
12             print(f'Contenido del archivo {nombre_archivo}:')
13             print(contenido)
14     except FileNotFoundError:
15         print(f'El archivo {nombre_archivo} no se encontró')
16
17 # Agregar contenido a un archivo existente
18 def agregar_contenido(nombre_archivo, contenido):
19     with open(nombre_archivo, 'a') as archivo:
20         archivo.write(contenido)
21     print(f'Se ha agregado contenido al archivo {nombre_archivo}')
22
23 nombre_archivo = 'ejemplo.txt'
24 escribir_archivo(nombre_archivo, 'Hola, este es un archivo de ejemplo.\n')
25 leer_archivo(nombre_archivo)
26 agregar_contenido(nombre_archivo, 'Esto es contenido adicional.\n')
27 leer_archivo(nombre_archivo)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Documentos\Algoritmos\Deberes\Deber 11> & C:/Users/josug/AppData/Local/Microsoft/WindowsApp
o_8_EjemploConsulta.py"
Se ha escrito el contenido en el archivo ejemplo.txt
Contenido del archivo ejemplo.txt:
Hola, este es un archivo de ejemplo.

Se ha agregado contenido al archivo ejemplo.txt
Contenido del archivo ejemplo.txt:
Hola, este es un archivo de ejemplo.
Esto es contenido adicional.

PS E:\Documentos\Algoritmos\Deberes\Deber 11> |
```

ENTREGABLES:

Una vez culminada tu tarea, capturar las pantallas de la ejecución del problema con tus datos y súbela en el apartado del aula virtual “S12-Deber-5: Tuplas, listas, funciones”

Subir los ejercicios al git o al drive y entrega la url de los archivos .py o, a su vez, entregue el archivo.

1. Recordar que el nombre del archivo deberá ser: **S12-Deber-5-2023B-NApellido**(de todos los integrantes)

RECURSOS NECESARIOS

- Acceso a Internet.
- Imaginación.
- VSC

RECOMENDACIÓN

Para optimizar el desarrollo en Python, es crucial dominar el manejo de arreglos, funciones y archivos. Se recomienda practicar regularmente la manipulación de arreglos para familiarizarse con las diferentes operaciones disponibles, como la indexación, el corte y la manipulación de elementos. Además, es importante comprender profundamente cómo funcionan las funciones en Python y cómo pasar parámetros por tuplas y listas para aprovechar al máximo su potencial. Asimismo, se aconseja practicar el manejo de archivos mediante la lectura y escritura de datos en diferentes formatos, como texto y binario. Dominar estos conceptos y técnicas permitirá a los desarrolladores crear aplicaciones más sólidas y eficientes en Python.

CONCLUSIÓN

El manejo efectivo de arreglos, funciones y archivos en Python es esencial para el desarrollo de aplicaciones eficientes y robustas. Los arreglos ofrecen una forma conveniente de almacenar y manipular datos de manera estructurada, lo que facilita la gestión de conjuntos de información. Las funciones permiten modularizar el código, promoviendo la reutilización y facilitando el mantenimiento del programa. Además, el paso de parámetros por tuplas y listas en funciones brinda flexibilidad y potencia para trabajar con datos dinámicos y complejos. Por último, el uso de archivos en Python proporciona la capacidad de leer y escribir datos de manera persistente, lo que es fundamental para aplicaciones que requieren almacenamiento de información a largo plazo.

ENLACES

Enlace GitHub:

<https://github.com/JosueGuerra2023B/Estructuras-Datos2023B/tree/master/Deberes/Deber%2011>

1 Bibliografía

- [1] T. School, «Esteban Canle Fernández,» 3 Mayo 2022. [En línea]. Available: <https://www.tokioschool.com/noticias/manejo-archivos-python/#%c2%bfque-es-el-manejo-de-archivos-en-python>. [Último acceso: 7 Febrero 2024].