

UNIVERSIDAD PRIVADA DOMINGO SAVIO

FACULTAD DE INGENIERIA



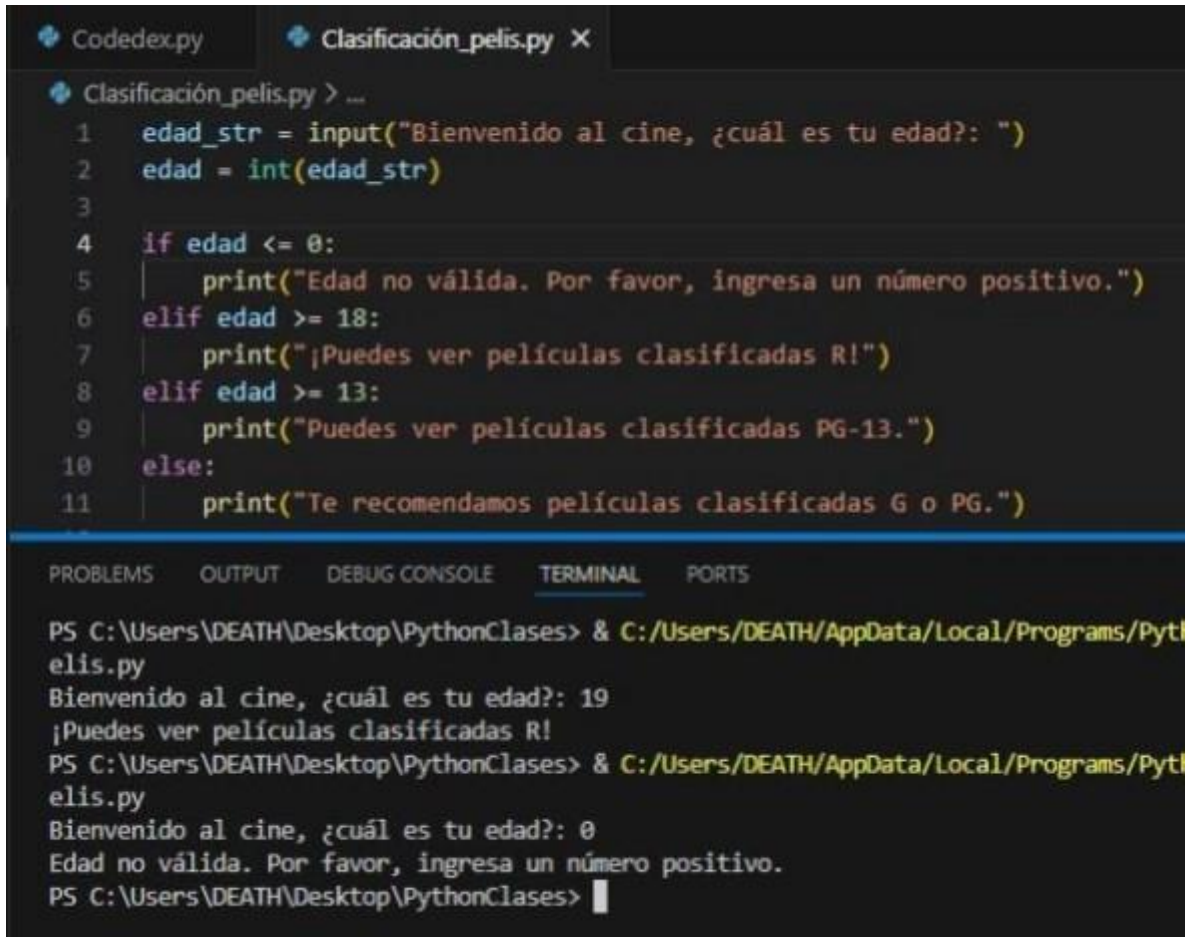
ACTIVIDAD 01 – PROGRAMACIÓN II

Docente: Ing. Jimmy Nataniel Requena Llorentty

Estudiante: Marco Aurelio Barriga

Ejercicios en clases – Programación II

Clasificación de películas



```

Codedex.py  Clasificación_pelis.py X
Clasificación_pelis.py > ...
1  edad_str = input("Bienvenido al cine, ¿cuál es tu edad?: ")
2  edad = int(edad_str)
3
4  if edad <= 0:
5      print("Edad no válida. Por favor, ingresa un número positivo.")
6  elif edad >= 18:
7      print("¡Puedes ver películas clasificadas R!")
8  elif edad >= 13:
9      print("Puedes ver películas clasificadas PG-13.")
10 else:
11     print("Te recomendamos películas clasificadas G o PG.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python39-64/Python.exe C:\Users\DEATH\Desktop\PythonClases\Clasificación_pelis.py
Bienvenido al cine, ¿cuál es tu edad?: 19
¡Puedes ver películas clasificadas R!
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python39-64/Python.exe C:\Users\DEATH\Desktop\PythonClases\Clasificación_pelis.py
Bienvenido al cine, ¿cuál es tu edad?: 0
Edad no válida. Por favor, ingresa un número positivo.
PS C:\Users\DEATH\Desktop\PythonClases>

```

Este programa es un sencillo verificador de edad para clasificar qué tipo de películas puede ver una persona en el cine, basado en su edad. Utiliza entrada del usuario, conversión de datos y estructuras condicionales if-elif-else para determinar la categoría de clasificación adecuada: R, PG-13, o G/PG. Además, incluye una validación básica para evitar edades no válidas (cero o negativas), lo que mejora su confiabilidad.

Bucles for-while

```
e.py
Contador
0
1
2
3

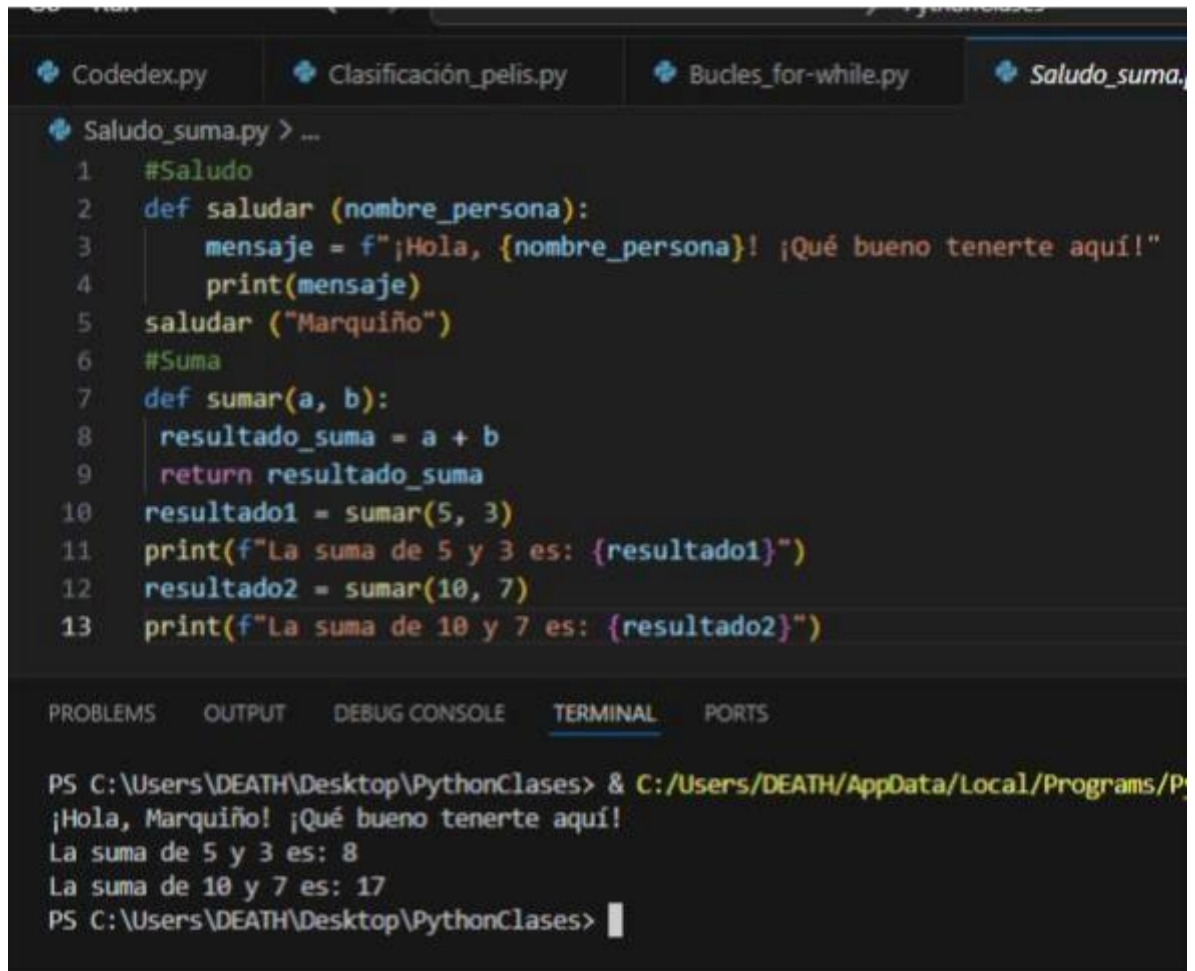
Recorriendo un string
M
a
r
q
u
i
ñ
o
Ingresa cualquier número: 5
tabla del 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

Bucle while
contador es: 0
contador es: 1
contador es: 2
contador es: 3
¡Bulce While terminado!
PS C:\Users\DEATH\Desktop\Pytho
```

```
ledex.py  Clasificación_pelis.py  Bucles_for-while.py
des_for-while.py > ...
#Contador
print ("Contador")
for numero in range (4):
    print (numero)
#Nombres
print ("\n Recorriendo un string")
nombre = "Marquiño"
for letra in nombre:
    print(letra)
#Multiplicación
tabla = int(input("Ingresa cualquier número: "))
print(f"tabla del {tabla}")
for i in range (1, 11):
    resultado = tabla * i
    print(f"{tabla} * {i} = {resultado}")
#While
contador = 0
print("\nBucle while")
while contador < 4:
    print(f"contador es: {contador}")
    contador = contador + 1
print("¡Bulce While terminado! ")
```

En este código Utiliza diferentes tipos de bucles (for y while), manejo de cadenas, entrada de usuario y operaciones aritméticas básicas. Cada sección cumple una función distinta, pero en conjunto muestran cómo automatizar tareas repetitivas, interactuar con el usuario y recorrer tanto rangos numéricos como cadenas de texto.

Saludar personas – Definición suma



```
Saludo_suma.py > ...
1  #Saludo
2  def saludar (nombre_persona):
3      mensaje = f"¡Hola, {nombre_persona}! ¡Qué bueno tenerte aquí!"
4      print(mensaje)
5      saludar ("Marquiño")
6  #Suma
7  def sumar(a, b):
8      resultado_suma = a + b
9      return resultado_suma
10 resultado1 = sumar(5, 3)
11 print(f"La suma de 5 y 3 es: {resultado1}")
12 resultado2 = sumar(10, 7)
13 print(f"La suma de 10 y 7 es: {resultado2}")

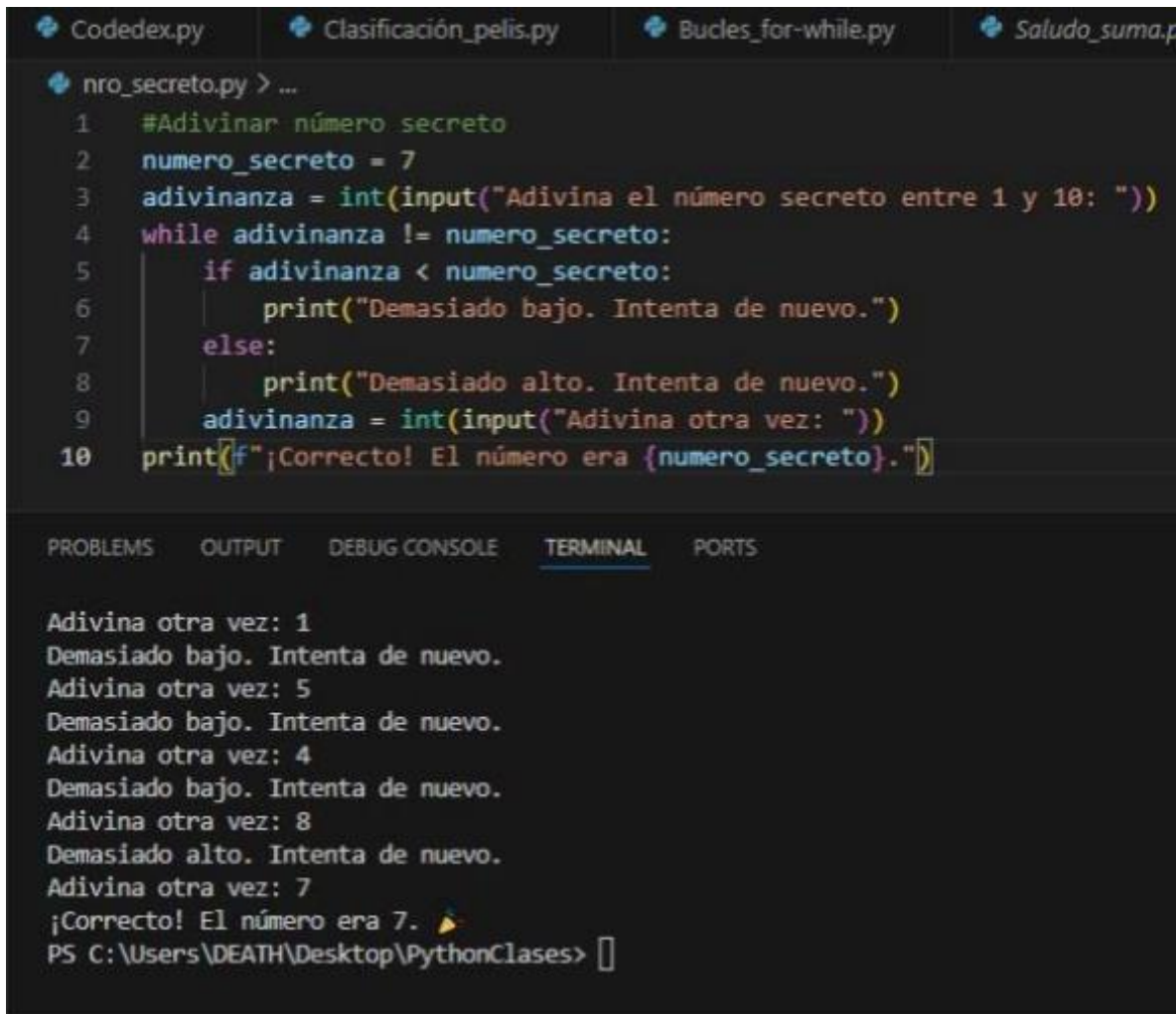
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python39-64/python.exe Saludo_suma.py
¡Hola, Marquiño! ¡Qué bueno tenerte aquí!
La suma de 5 y 3 es: 8
La suma de 10 y 7 es: 17
PS C:\Users\DEATH\Desktop\PythonClases> 
```

El primer código muestra cómo crear una función que genera un mensaje personalizado. Es un ejemplo simple pero útil para entender cómo funcionan los parámetros y el uso de fstrings.

El segundo código realiza una suma entre dos números y devuelve el resultado. Se utiliza dos veces con distintos valores, mostrando cómo reutilizar funciones para tareas repetitivas.

Adivinar número secreto



```
CodeDex.py Clasificación_pelis.py Bucles_for-while.py Saludo_suma.p
nro_secreto.py > ...
1 #Adivinar número secreto
2 numero_secreto = 7
3 adivinanza = int(input("Adivina el número secreto entre 1 y 10: "))
4 while adivinanza != numero_secreto:
5     if adivinanza < numero_secreto:
6         print("Demasiado bajo. Intenta de nuevo.")
7     else:
8         print("Demasiado alto. Intenta de nuevo.")
9     adivinanza = int(input("Adivina otra vez: "))
10 print(f"¡Correcto! El número era {numero_secreto}.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Adivina otra vez: 1
Demasiado bajo. Intenta de nuevo.
Adivina otra vez: 5
Demasiado bajo. Intenta de nuevo.
Adivina otra vez: 4
Demasiado bajo. Intenta de nuevo.
Adivina otra vez: 8
Demasiado alto. Intenta de nuevo.
Adivina otra vez: 7
¡Correcto! El número era 7. 🎉
PS C:\Users\DEATH\Desktop\PythonClases> []
```

En esta parte del código utiliza una estructura while para repetir el intento hasta que el usuario acierte. Las condicionales if-else para dar pistas ("Demasiado bajo" o "Demasiado alto").

Entrada del usuario con input () convertida a entero, lo que permite comparar correctamente con el número secreto. Es un código donde se utiliza bucles, condicionales y entradas de usuario

Verificador de películas

```
Verificación_pelis.py > main
1  # Ejercicio: Verificador de Edad para Películas usando Funciones
2  def obtener_clasificacion_pelicula(edad_persona):
3      if edad_persona < 0 or edad_persona > 120:

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== VERIFICADOR DE EDAD PARA PELÍCULAS ===

Por favor, Ingresa tu edad: 19

Resultado: ¡Puedes ver películas de cualquier clasificación incluyendo R!

¡Gracias por usar nuestro verificador!

¿Quieres ver las pruebas con diferentes edades? (s/n): s

=== PRUEBAS CON DIFERENTES EDADES ===
Edad 5: Te recomendamos películas G (General Audiencias).
Edad 12: Te recomendamos películas G (General Audiencias).
Edad 13: Puedes ver películas G y PG-13.
Edad 16: Puedes ver películas G y PG-13.
Edad 17: ¡Puedes ver películas de cualquier clasificación incluyendo R!
Edad 25: ¡Puedes ver películas de cualquier clasificación incluyendo R!
Edad 65: ¡Puedes ver películas de cualquier clasificación incluyendo R!
Edad -5: Edad no válida.
Edad 150: Edad no válida.
PS C:\Users\DEATH\Desktop\PythonClases> |
```

En este código se determina la clasificación de películas adecuada según la edad. Incluye validación de datos y condiciones claras, lo cual mejora la confiabilidad. Es la función principal del programa. Se encarga de interactuar con el usuario, pedir su edad, y mostrar el resultado de la clasificación.

Ejemplo de refactorización

```
Refactorizacion.py > mostrar_area_rectangulo
1  # Ejemplo de codigo a refactorizar:
2  # Rectangulo 1
3  base1 = 10
4  altura1 = 5
5  area1 = base1*altura1
6  print(f"El area del rectangulo 1 ({base1} x {altura1}) es:{area1} ")
7
8  #Rectangulo 2
9  base2 = 7
10 altura2 = 3
11 area2 = base2 * altura2
12 print (f"El area del rectangulo2 ({base2} * {altura2}) es: {area2}")
13
14 def mostrar_area_rectangulo(numero,base,altura):
15     area = base * altura
16     print(f"El area del rectangulo {numero} ({base}x{altura}) es: {area}")
```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python39-64/Python.exe Refactorizacion.py
El area del rectangulo 1 (10 x 5) es:50
El area del rectangulo2 (7 * 3) es: 21
PS C:\Users\DEATH\Desktop\PythonClases>
```

Este código muestra cómo calcular e imprimir el área de rectángulos. Inicialmente, el cálculo se realiza de forma repetitiva para dos rectángulos, repitiendo la misma lógica con diferentes variables. Luego, se incluye una función llamada `mostrar_area_rectangulo`, que encapsula esa lógica y permite reutilizar el código de manera más limpia y eficiente.

Notas parciales

```
parciales.py > ...
1  notas_parciales = [80, 95, 73, 60, 88]
2  primera_nota = notas_parciales[0]
3  print(f"La primera nota fue: {primera_nota}")
4  tercera_nota = notas_parciales[2]
5  print(f"La tercera nota fue: {tercera_nota}")
6  print(f"Lista original: {notas_parciales}")
7  notas_parciales[3] = 65
8  print(f"Lista modificada: {notas_parciales}")
9  cantidad_de_notas = len(notas_parciales)
10 print(f"Tenemos un total de {cantidad_de_notas} notas.")

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local
La primera nota fue: 80
La tercera nota fue: 73
Lista original: [80, 95, 73, 60, 88]
Lista modificada: [80, 95, 73, 65, 88]
Tenemos un total de 5 notas.
PS C:\Users\DEATH\Desktop\PythonClases>
```

El código accede a notas específicas dentro de una lista, modifica un valor, imprime la lista antes y después del cambio, y finalmente muestra cuántas notas hay en total.

Lista de hobbies-comidas favoritas

```
hobbies.py > ...
1  #Hobbies
2  hobbies_M = ["Dibujar", "Jugar juegos", "Ver series, dibujos"]
3  lista_hobbies = hobbies_M
4  print(f"Estos son mis hobbies: {lista_hobbies}")
5  print(f"Mi segundo hobbies es: {lista_hobbies[1]}")
6  comidas_favoritas = ["Picante de pollo", "Pollo a la broaster", "Sopa de mani" ]
7  print(f"Estas son mis comidas favoritas: {comidas_favoritas}")
8  print(f"Mi segunda comida cambiara de: {comidas_favoritas[1]}")
9  comidas_favoritas[1] = "Milanesa de carne"
10 print(f"Mi segunda comida cambiara a: {comidas_favoritas[1]}")
11 print(f"Ahora estas son mis comidas favoritas: {comidas_favoritas}")
12

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python313/py
Estos son mis hobbies: ['Dibujar', 'Jugar juegos', 'Ver series, dibujos']
Mi segundo hobbies es: Jugar juegos
Estas son mis comidas favoritas: ['Picante de pollo', 'Pollo a la broaster', 'Sopa de mani']
Mi segunda comida cambiara de: Pollo a la broaster
Mi segunda comida cambiara a: Milanesa de carne
Ahora estas son mis comidas favoritas: ['Picante de pollo', 'Milanesa de carne', 'Sopa de mani']
PS C:\Users\DEATH\Desktop\PythonClases>
```

Este código trabaja con listas en Python para almacenar y manipular datos personales como hobbies y comidas favoritas. Es un ejemplo básico de el uso de listas en Python. **Lista de integrantes Death Services**

```
Death_services.py
1  #Lista de programación II
2  #Death Services
3  Death_services = ["Adaniel Balderrama Orellana", "Marco Aurelio Barriga", "Carlos Ademar Suarez"]
4  for nombre in Death_services:
5      print(f"¡Bienvenido al grupo {nombre}!")

Ln 5, Col 45 • Spaces: 4 Histo

> Console Shell x +
~/workspace: python Death_services.py 2

~/workspace$ python Death_services.py
¡Bienvenido al grupo Adaniel Balderrama Orellana!
¡Bienvenido al grupo Marco Aurelio Barriga!
¡Bienvenido al grupo Carlos Ademar Suarez!
~/workspace$
```

Este código muestra cómo trabajar con una lista de cadenas y un bucle for para recorrer sus elementos. En este caso, se crea una lista llamada Death_services que contiene nombres

de integrantes de un grupo. Luego, con un bucle for, se itera por cada nombre en la lista y se imprime un mensaje personalizado de bienvenida usando una f-string.

Suma de notas

```
mis_notas.py > ...
1  notas = [85.5, 92, 78, 88.5, 95, 82]
2  suma_t = 0
3  for nota in notas:
4      suma_t += nota
5  promedio = suma_t / len(notas)
6  print(f"Suma total de las notas: {suma_t}")
7  print(f"Promedio de las notas: { promedio:.2f} ")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppDa
Suma total de las notas: 521.0
Promedio de las notas: 86.83
PS C:\Users\DEATH\Desktop\PythonClases>
```

Este código utiliza un bucle for para que recorra cada numero o elemento que se le asigno, se le asigna una lista de notas para que puedan ser calculadas y nos de como resultado la suma total y así obtengamos un promedio total de toda la función.

Clase 06 operaciones listas

Algoritmo 1

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python38-32/Python.exe C:\Users\DEATH\Desktop\PythonClases\sumar_elementos.py
Probando sumar_elementos...
¡Pruebas para sumar_elementos pasaron! ✓

--- Pruebas adicionales ---
Suma de [1, 2, 3, 4, 5]: 15
Suma de [10, -2, 5]: 13
Suma de lista vacía []: 0
Suma de [100]: 100
Suma de [-1, -2, -3]: -6
Suma de [0, 0, 0, 0]: 0

Marco Aurelio - FIN DEL PROGRAMA SUMA DE ELEMENTOS
PS C:\Users\DEATH\Desktop\PythonClases>
```

Algoritmo 2

```
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python38-32/Python.exe C:\Users\DEATH\Desktop\PythonClases\encontrar_mayor.py
Probando encontrar_mayor...

Lista: [1, 9, 2, 8, 3, 7] -> Mayor encontrado: 9
Lista: [-1, -9, -2, -8] -> Mayor encontrado: -1
Lista: [42, 42, 42] -> Mayor encontrado: 42
Lista: [] -> Mayor encontrado: None
Lista: [5] -> Mayor encontrado: 5

¡Pruebas para encontrar_mayor pasaron! ✓

Marco Aurelio - FIN DEL PROGRAMA ENCUENTRA EL MAYOR ELEMENTO
PS C:\Users\DEATH\Desktop\PythonClases>
```

Algoritmo 3

```
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python313/python.exe
Probando contar_apariciones...

Elemento '4' aparece 5 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '2' aparece 2 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '9' aparece 0 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '7' aparece 1 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]

Marco Aurelio- FIN DEL PROGRAMA QUE CUENTA LAS VECES QUE APARECE UN ELEMENTO
PS C:\Users\DEATH\Desktop\PythonClases>
```

Algoritmo 4

```
PS C:\Users\DEATH\Desktop\PythonClases> & C:/Users/DEATH/AppData/Local/Programs/Python/Python313/python.exe
Probando invertir_lista...

Lista original: [1, 2, 3, 4, 5]
Lista invertida: [5, 4, 3, 2, 1]
¡Pruebas para invertir_lista pasaron! ✓

Marco Aurelio - FIN DEL PROGRAMA QUE INVIERTE LOS ELEMENTOS DE UNA LISTA
PS C:\Users\DEATH\Desktop\PythonClases>
```

El código implementa cuatro funciones fundamentales para operar con listas en Python, cada una con su lógica específica, pruebas con assert y ejemplos de uso. Nos da como resultado todas las pruebas que recorre cada función de cada una de los algoritmos, suma de elementos, encontrar el mayor, contar apariciones y una lista invertida, todos nos imprimen como resultado que pasaron la prueba.

Búsqueda Lineal-Binaria

```
mis_notas.py  Verificación_pelis.py  Saludo_suma.py  acumulador_suma.p
C: > Users > DEATH > Downloads > clase06_búsquedas.py > ...
1  # clase06_búsquedas.py
2  # 1. Definir la función búsqueda_lineal
3  def búsqueda_lineal(lista, clave):
4      for i in range(len(lista)):
5          if lista[i] == clave:
6              return i
7      return -1
8
9  # 3. Prueba tu función con assert
10 mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
11 print("Probando búsqueda_lineal...")
12
13 assert búsqueda_lineal(mi_lista_desordenada, 42) == 2
14 assert búsqueda_lineal(mi_lista_desordenada, 10) == 0 # Al inicio

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

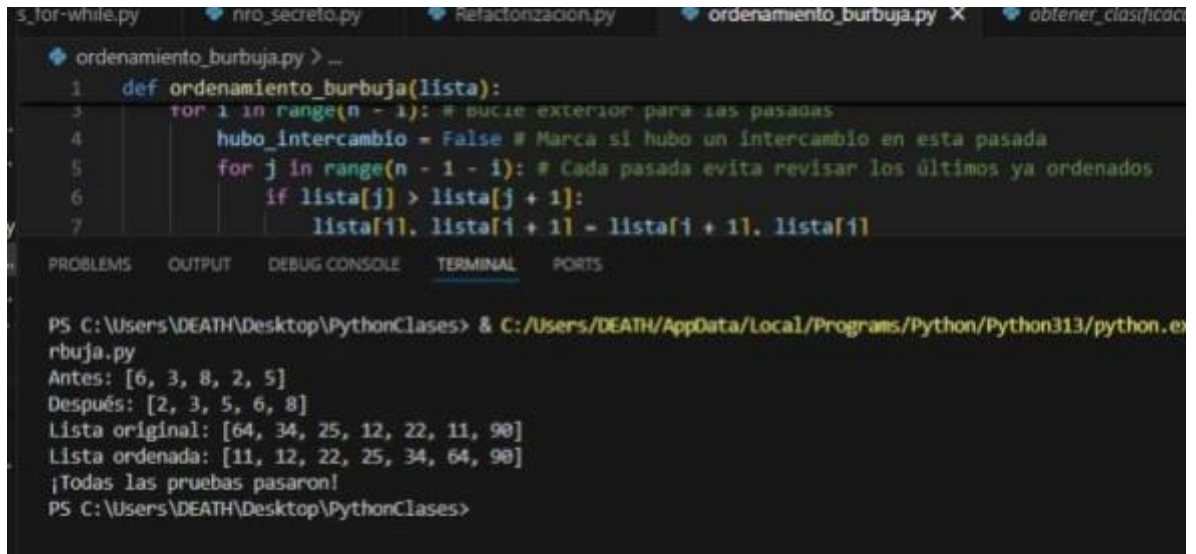
¡Pruebas para búsqueda_lineal pasaron! ✓
Marco Aurelio - FIN DEL PROGRAMA

Probando búsqueda_binaria...
¡Pruebas para búsqueda_binaria pasaron! ✓
Marco Aurelio - FIN DEL PROGRAMA

Realizando el experimento del caos...
Búsqueda binaria de '5' en lista desordenada devolvió: 1
Marco Aurelio - FIN DEL PROGRAMA
PS C:\Users\DEATH\Desktop\PythonClases>
```

Busca un elemento (clave) en una lista y devuelve su posición (índice) si lo encuentra, o 1 si no está presente. El **bucle** for: Recorre cada índice de la lista con `range(len(lista))` en la comparación verifica si el elemento en la posición actual (`lista[i]`) es igual a clave, si coincide, retorna el índice `i` inmediatamente (termina la función). Si el bucle termina sin encontrar clave, retorna -1.

Ordenamiento burbuja



```
ordenamiento_burbuja.py > ...
1 def ordenamiento_burbuja(lista):
2     for i in range(n - 1): # bucle exterior para las pasadas
3         hubo_intercambio = False # Marca si hubo un intercambio en esta pasada
4         for j in range(n - 1 - i): # Cada pasada evita revisar los últimos ya ordenados
5             if lista[j] > lista[j + 1]:
6                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
7     return lista

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\DEATH\Desktop\PythonClases> & C:\Users\DEATH\AppData\Local\Programs\Python\Python313\python.exe ordenamiento_burbuja.py
Antes: [6, 3, 8, 2, 5]
Después: [2, 3, 5, 6, 8]
Lista original: [64, 34, 25, 12, 22, 11, 90]
Lista ordenada: [11, 12, 22, 25, 34, 64, 90]
¡Todas las pruebas pasaron!
PS C:\Users\DEATH\Desktop\PythonClases>
```

Este código implementa el algoritmo de ordenamiento de burbuja (Bubble Sort), ordena listas eficientemente, para tamaños pequeños, controla el número de pasadas necesarias para ordenar la lista, cada pasada coloca el elemento más grande no ordenado en su posición correcta al final de la lista. Compara pares de elementos adyacentes ($lista[j]$ y $lista[j + 1]$).

Si están en el orden incorrecto ($lista[j] > lista[j + 1]$), los intercambia, Si en una pasada completa **no hubo intercambios**, significa que la lista ya está ordenada. El algoritmo termina antes (break) para evitar pasadas innecesarias.

```
function myFunction() {  
    DocumentApp.getUi()  
        .createMenu(' @ Mis Funciones')  
        .addItem(' Insertar Fecha y Hora', 'insertarFechaYHora')  
        .addToUi();  
}
```

```
function insertarFechaYHora() {    var body =  
DocumentApp.getActiveDocument().getBody();  
    var fecha = Utilities.formatDate(new Date(), Session.getScriptTimeZone(),  
"dd/MM/yyyy HH:mm:ss"); body.appendParagraph(fecha);  
}
```

17/06/2025 08:17:01