

UNIVERSIDAD PRIVADA DOMINGO SAVIO

FACULTAD DE INGENIERIA



Actividad

Título: Actividad 01

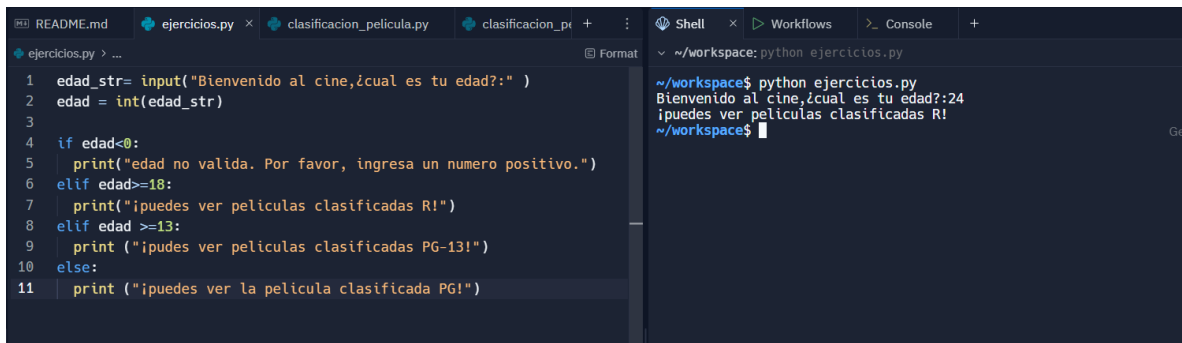
Ejercicios prácticos programación

Docente: Ing.Jimmy Nataniel Requena Llorentty

Materia: Programación 2

Estudiante: Sarai Alejandra Vidaurre

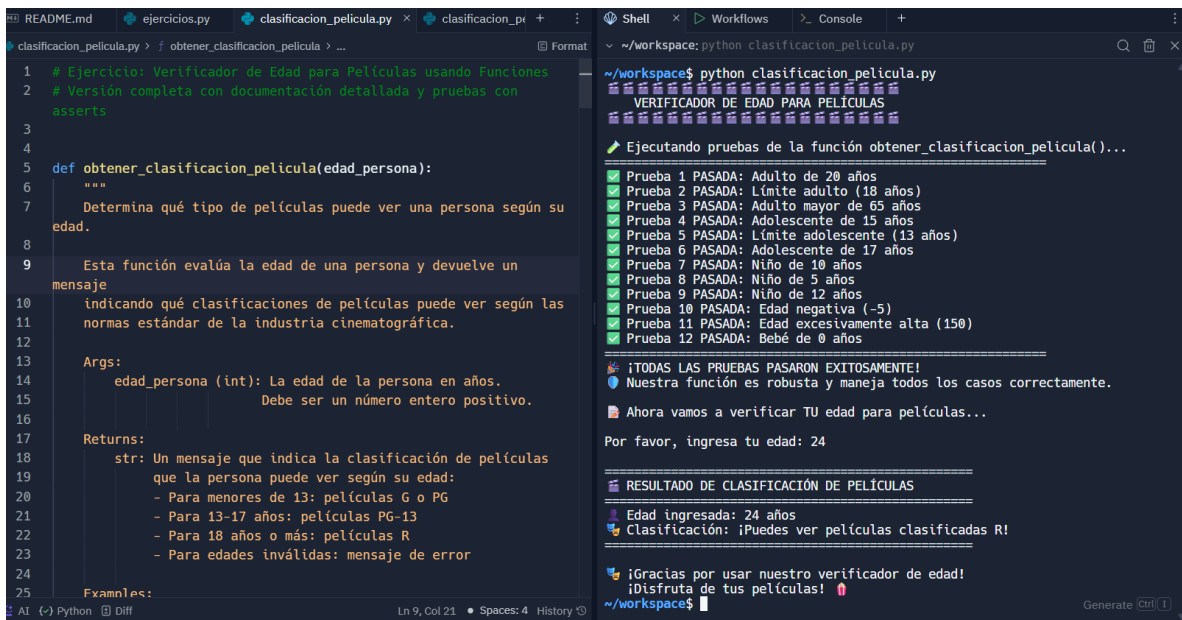
Junio del 2025
Santa Cruz – Bolivia



```
1 edad_str= input("Bienvenido al cine,¿cual es tu edad?:" )
2 edad = int(edad_str)
3
4 if edad<0:
5     print("edad no valida. Por favor, ingresa un numero positivo.")
6 elif edad>=18:
7     print("¡puedes ver peliculas clasificadas R!")
8 elif edad >=13:
9     print ("¡pudes ver peliculas clasificadas PG-13!")
10 else:
11     print ("¡puedes ver la pelicula clasificada PG!")
```

```
~/workspace$ python ejercicios.py
Bienvenido al cine,¿cual es tu edad?:24
¡puedes ver peliculas clasificadas R!
~/workspace$
```

Ilustración 1 ejercicio #1



```
1 # Ejercicio: Verificador de Edad para Películas usando funciones
2 # Versión completa con documentación detallada y pruebas con
3 # asserts
4
5 def obtener_clasificacion_pelicula(edad_persona):
6     """
7     Determina qué tipo de películas puede ver una persona según su
8     edad.
9
10    Esta función evalúa la edad de una persona y devuelve un
11    mensaje
12    indicando qué clasificaciones de películas puede ver según las
13    normas estándar de la industria cinematográfica.
14
15    Args:
16        edad_persona (int): La edad de la persona en años.
17        Debe ser un número entero positivo.
18
19    Returns:
20        str: Un mensaje que indica la clasificación de películas
21        que la persona puede ver según su edad:
22        - Para menores de 13: películas G o PG
23        - Para 13-17 años: películas PG-13
24        - Para 18 años o más: películas R
25        - Para edades inválidas: mensaje de error
26
27    Examples:
28        obtener_clasificacion_pelicula(10) returns 'G'
29        obtener_clasificacion_pelicula(15) returns 'PG-13'
30        obtener_clasificacion_pelicula(20) returns 'R'
31        obtener_clasificacion_pelicula(-5) returns 'Error: Edad inválida'
32        obtener_clasificacion_pelicula(150) returns 'Error: Edad inválida'
33    """
```

```
~/workspace$ python clasificacion_pelicula.py
VERIFICADOR DE EDAD PARA PELÍCULAS
Ejecutando pruebas de la función obtener_clasificacion_pelicula()...
✓ Prueba 1 PASADA: Adulto de 20 años
✓ Prueba 2 PASADA: Límite adulto (18 años)
✓ Prueba 3 PASADA: Adulto mayor de 65 años
✓ Prueba 4 PASADA: Adolescente de 15 años
✓ Prueba 5 PASADA: Límite adolescente (13 años)
✓ Prueba 6 PASADA: Adolescente de 17 años
✓ Prueba 7 PASADA: Niño de 10 años
✓ Prueba 8 PASADA: Niño de 5 años
✓ Prueba 9 PASADA: Niño de 12 años
✓ Prueba 10 PASADA: Edad negativa (-5)
✓ Prueba 11 PASADA: Edad excesivamente alta (150)
✓ Prueba 12 PASADA: Bebé de 0 años
¡TODAS LAS PRUEBAS PASARON EXITOSAMENTE!
Nuestra función es robusta y maneja todos los casos correctamente.
Ahora vamos a verificar TU edad para películas...
Por favor, ingresa tu edad: 24
RESULTADO DE CLASIFICACIÓN DE PELÍCULAS
Edad ingresada: 24 años
Clasificación: ¡Puedes ver películas clasificadas R!
¡Gracias por usar nuestro verificador de edad!
¡Disfruta de tus películas!
```

Ilustración 2 ejercicio clasificación de película con asserts

```
ejercicios.py clasificacion_pelicula.py clasificacion_pelicula_corto.py + : Shell x Workflows >_ Console +
clasificacion_pelicula_corto.py > ... Format ~ /workspace: python clasificacion_pelicula_corto.py
1 # Ejercicio: Verificador de Edad para Películas usando Funciones
2
3 def obtener_clasificacion_pelicula(edad_persona):
4     """
5     Función que determina qué tipo de películas puede ver una
6     persona según su edad.
7
8     Parámetro:
9         edad_persona (int): La edad de la persona
10
11     Retorna:
12         str: Mensaje con la clasificación de película recomendada
13     """
14
15     # Validar que la edad sea válida
16     if edad_persona < 0 or edad_persona > 120:
17         return "Edad no válida."
18
19     # Lógica de clasificación por edad
20     if edad_persona < 13:
21         return "Te recomendamos películas G (General Audiences).".
22     elif edad_persona < 17:
23         return "Puedes ver películas G y PG-13."
24     elif edad_persona >= 17:
25         return "¡Puedes ver películas de cualquier clasificación incluyendo R!"
26
27 ~ /workspace$ python clasificacion_pelicula_corto.py
28 == VERIFICADOR DE EDAD PARA PELÍCULAS ==
29
30 Por favor, ingresa tu edad: 24
31
32 Resultado: ¡Puedes ver películas de cualquier clasificación incluyendo R!
33
34 ¡Gracias por usar nuestro verificador!
35
36 ¿Quieres ver las pruebas con diferentes edades? (s/n): s
37
38 == PRUEBAS CON DIFERENTES EDADES ==
39 Edad 5: Te recomendamos películas G (General Audiences).
40 Edad 12: Te recomendamos películas G (General Audiences).
41 Edad 13: Puedes ver películas G y PG-13.
42 Edad 16: Puedes ver películas G y PG-13.
43 Edad 17: ¡Puedes ver películas de cualquier clasificación incluyendo R!
44 Edad 25: ¡Puedes ver películas de cualquier clasificación incluyendo R!
45 Edad 65: ¡Puedes ver películas de cualquier clasificación incluyendo R!
46 Edad -5: Edad no válida.
47 Edad 150: Edad no válida.
48 ~ /workspace$
```

Ilustración 3 Clasificación de película con funciones

```
clasificacion_pelicula_corto.py tabla_de_multiplicar.py x bucle_for.py + : Shell x Workflows >_ Console +
tabla_de_multiplicar.py > ... Format ~ /workspace: python tabla_de_multiplicar.py
1 num_tabla = int(input( "introduce el numero de la tabla: " ))
2 print(f"--- Tabla del {num_tabla} ---")
3 for i in range(1, 11):
4     resultado = num_tabla * i
5     print(f"{num_tabla} x {i} = {resultado}")
6
7 ~ /workspace$ python tabla_de_multiplicar.py
8 introduce el numero de la tabla: 2
9 --- Tabla del 2 ---
10 2 x 1 = 2
11 2 x 2 = 4
12 2 x 3 = 6
13 2 x 4 = 8
14 2 x 5 = 10
15 2 x 6 = 12
16 2 x 7 = 14
17 2 x 8 = 16
18 2 x 9 = 18
19 2 x 10 = 20
20 ~ /workspace$
```

Ilustración 4 Tabla de multiplicar

```
de_multiplicar.py bucle_for.py x bucle_while.py lista_mis_hobbies.py + : Shell x Workflows >_ Console +
bucle_for.py > ... Format ~ /workspace: python bucle_for.py
1 print("Contando hasta 3 (sin incluirlo):")
2 for numero in range(3):
3     print(numero)
4 print("\nRecorriendo un string: ")
5 nombre= "PYTHON"
6 for letra in nombre:
7     print(letra)
8
9 ~ /workspace$ python bucle_for.py
10 Contando hasta 3 (sin incluirlo):
11 0
12 1
13 2
14
15 Recorriendo un string:
16 P
17 Y
18 T
19 H
20 O
21 N
22 ~ /workspace$
```

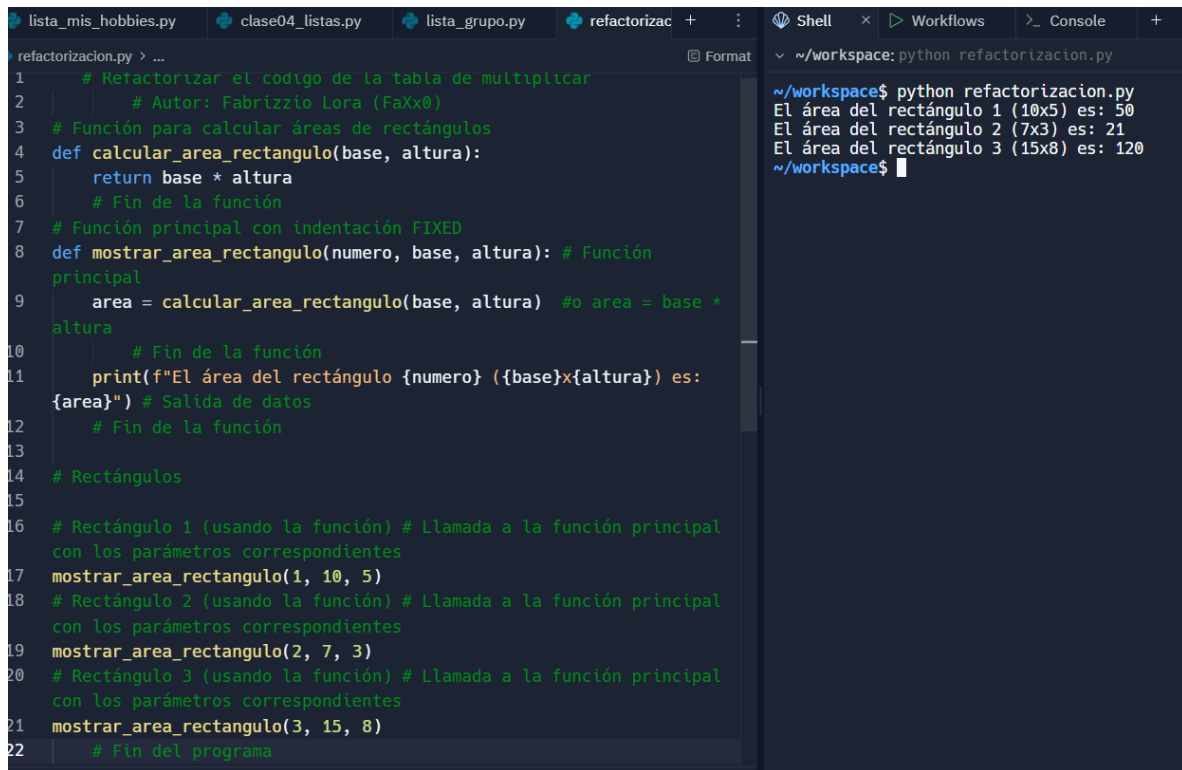
Ilustración 5 bucle for

```
le_multiplicar.py  bucle_for.py  bucle_while.py x  lista_mis_hobbies.py  +  :  Shell x  Workflows  >_ Console  +  
bucle_while.py > ...  Format  
1  contador = 0  
2  print("\nBucle while:")  
3  while contador < 3:  
4      print (f"Contador es: {contador}")  
5      contador = contador + 1  
6  print("¡Bucle while terminado!")  
  
~/workspace:python bucle_while.py  
~/workspace$ python bucle_while.py  
Bucle while:  
Contador es: 0  
¡Bucle while terminado!  
Contador es: 1  
¡Bucle while terminado!  
Contador es: 2  
¡Bucle while terminado!  
~/workspace$
```

Ilustración 6 bucle while

```
le_multiplicar.py  bucle_for.py  bucle_while.py  lista_mis_hobbies.py x  +  :  Shell x  Workflows  >_ Console  +  
lista_mis_hobbies.py > ...  Format  
1  notas_parciales = [80, 95, 73, 60, 88]  
2  primera_not = notas_parciales[0] # Índice 0 para el PRIMER elemento  
3  print(f"La primera nota fue: {primera_not}") # Imprimirá 80  
4  tercera_not = notas_parciales[2] # Índice 2 para el TERCER elemento  
5  print(f"La tercera nota fue: {tercera_not}") # Imprimirá 73  
6  
7  print(f"Lista original: {notas_parciales}")  
8  # Supongamos que se recalificó el 4to parcial (índice 3)  
9  notas_parciales[3] = 65 # Asignamos un nuevo valor al índice 3  
10 print(f"Lista modificada: {notas_parciales}")  
11  
12 cantidad_de_notas = len(notas_parciales)  
13 print(f"Tenemos un total de {cantidad_de_notas} notas.") #  
14     Imprimirá 5  
15 mis_hobbies = ["cantar", "jugar", "escribir"]  
16 print (f"Lista original: {mis_hobbies}" " Sarai, yo tengo 3 hobbies")  
  
~/workspace:python lista_mis_hobbies.py  
~/workspace$ python lista_mis_hobbies.py  
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.  
Lista original: ['cantar', 'jugar', 'escribir'] Sarai, yo tengo 3 hobbies  
~/workspace$
```

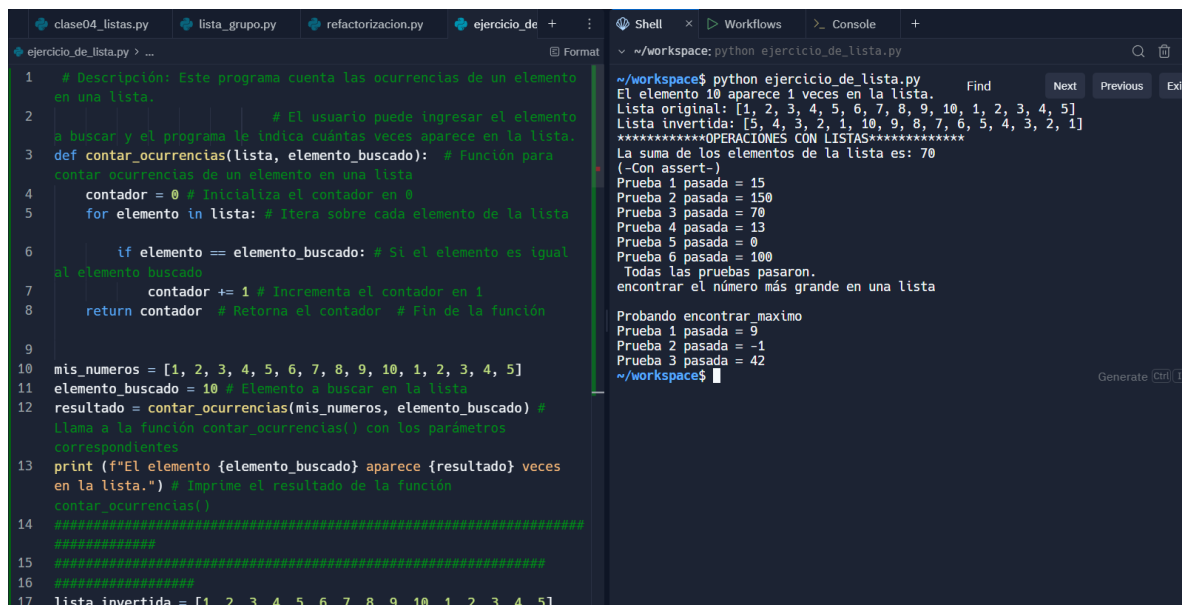
Ilustración 7 lista mis hobbies



```
lista_mis_hobbies.py  clase04_listas.py  lista_grupo.py  refactorizac  +  :  Shell  x  Workflows  >_ Console  +
refactorizacion.py > ...
1  # Refactorizar el código de la tabla de multiplicar
2  # Autor: Fabrizio Lora (FaXx0)
3  # Función para calcular áreas de rectángulos
4  def calcular_area_rectangulo(base, altura):
5      return base * altura
6      # Fin de la función
7  # Función principal con indentación FIXED
8  def mostrar_area_rectangulo(numero, base, altura): # Función principal
9      area = calcular_area_rectangulo(base, altura) #o area = base *
altura
10     # Fin de la función
11     print(f"El área del rectángulo {numero} ({base}x{altura}) es:
{area}") # Salida de datos
12     # Fin de la función
13
14 # Rectángulos
15
16 # Rectángulo 1 (usando la función) # Llamada a la función principal
con los parámetros correspondientes
17 mostrar_area_rectangulo(1, 10, 5)
18 # Rectángulo 2 (usando la función) # Llamada a la función principal
con los parámetros correspondientes
19 mostrar_area_rectangulo(2, 7, 3)
20 # Rectángulo 3 (usando la función) # Llamada a la función principal
con los parámetros correspondientes
21 mostrar_area_rectangulo(3, 15, 8)
22     # Fin del programa

~/workspace$ python refactorizacion.py
El área del rectángulo 1 (10x5) es: 50
El área del rectángulo 2 (7x3) es: 21
El área del rectángulo 3 (15x8) es: 120
~/workspace$
```

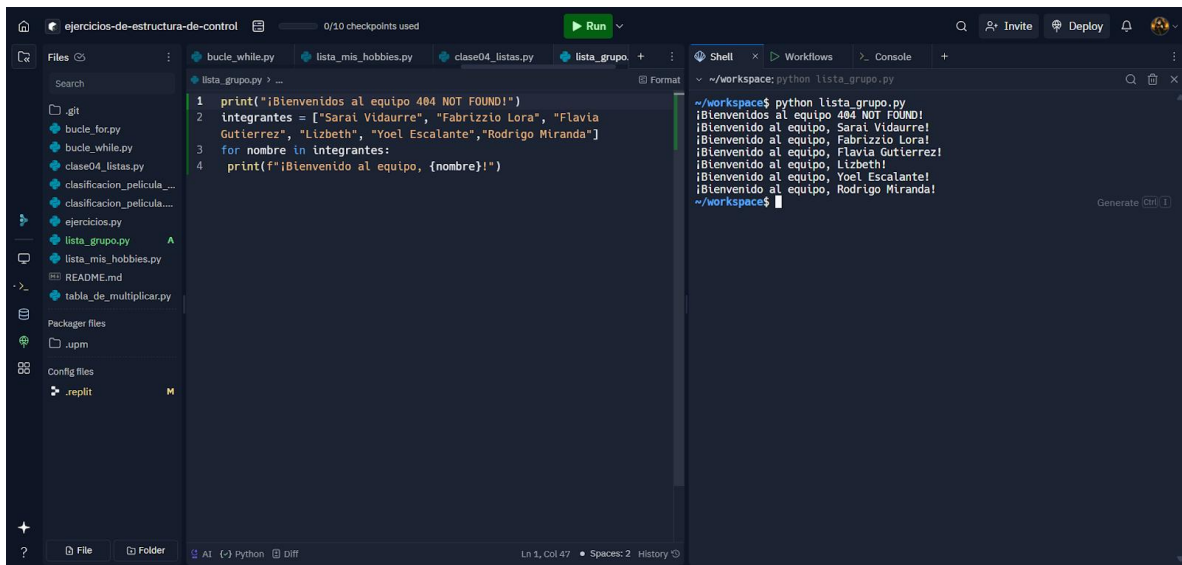
Ilustración 8 ejercicio de refactorización



```
clase04_listas.py  lista_grupo.py  refactorizacion.py  ejercicio_de  +  :  Shell  x  Workflows  >_ Console  +
ejercicio_de_lista.py > ...
1  # Descripción: Este programa cuenta las ocurrencias de un elemento
en una lista.
2  # El usuario puede ingresar el elemento
a buscar y el programa le indica cuántas veces aparece en la lista.
3  def contar_ocurrencias(lista, elemento_buscado): # Función para
contar ocurrencias de un elemento en una lista
4      contador = 0 # Inicializa el contador en 0
5      for elemento in lista: # Itera sobre cada elemento de la lista
6          if elemento == elemento_buscado: # Si el elemento es igual
al elemento buscado
7              contador += 1 # Incrementa el contador en 1
8      return contador # Retorna el contador # Fin de la función
9
10 mis_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
11 elemento_buscado = 10 # Elemento a buscar en la lista
12 resultado = contar_ocurrencias(mis_numeros, elemento_buscado) #
llama a la función contar_ocurrencias() con los parámetros
correspondientes
13 print(f"El elemento {elemento_buscado} aparece {resultado} veces
en la lista.") # Imprime el resultado de la función
contar_ocurrencias()
14
15
16
17 lista_invertida = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]

~/workspace$ python ejercicio_de_lista.py
El elemento 10 aparece 1 veces en la lista.
Lista original: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
Lista invertida: [5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
*****OPERACIONES CON LISTAS*****
La suma de los elementos de la lista es: 70
(-Con assert-)
Prueba 1 pasada = 15
Prueba 2 pasada = 150
Prueba 3 pasada = 70
Prueba 4 pasada = 13
Prueba 5 pasada = 0
Prueba 6 pasada = 100
Todas las pruebas pasaron.
encontrar el número más grande en una lista
Probando encontrar_maximo
Prueba 1 pasada = 9
Prueba 2 pasada = -1
Prueba 3 pasada = 42
~/workspace$
```

Ilustración 9 ejercicio de listas clase 05



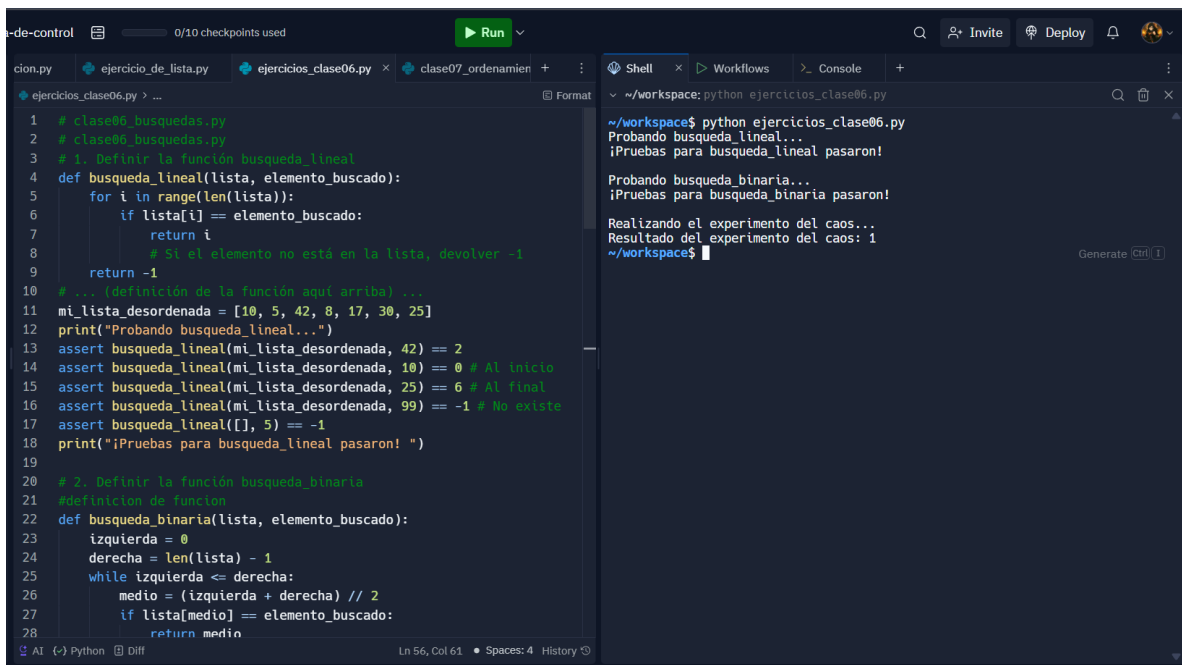
The screenshot shows a Replit workspace with a file explorer on the left containing several Python files. The main editor displays the code for 'lista_grupo.py':

```
1 print("¡Bienvenidos al equipo 404 NOT FOUND!")
2 integrantes = ["Sara Vidaurre", "Fabrizio Lora", "Flavia Gutierrez", "Lizbeth", "Yoel Escalante", "Rodrigo Miranda"]
3 for nombre in integrantes:
4     print(f"¡Bienvenido al equipo, {nombre}!")
```

The console on the right shows the output of running the script:

```
~/workspace$ python lista_grupo.py
¡Bienvenidos al equipo 404 NOT FOUND!
¡Bienvenido al equipo, Sara Vidaurre!
¡Bienvenido al equipo, Fabrizio Lora!
¡Bienvenido al equipo, Flavia Gutierrez!
¡Bienvenido al equipo, Lizbeth!
¡Bienvenido al equipo, Yoel Escalante!
¡Bienvenido al equipo, Rodrigo Miranda!
```

Ilustración 10 ejercicio grupal



The screenshot shows a Replit workspace with a file explorer on the left. The main editor displays the code for 'ejercicios_clase06.py':

```
1 # clase06_búsquedas.py
2 # clase06_búsquedas.py
3 # 1. Definir la función búsqueda_lineal
4 def búsqueda_lineal(lista, elemento_buscado):
5     for i in range(len(lista)):
6         if lista[i] == elemento_buscado:
7             return i
8         # Si el elemento no está en la lista, devolver -1
9     return -1
10 # ... (definición de la función aquí arriba) ...
11 mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
12 print("Probando búsqueda_lineal...")
13 assert búsqueda_lineal(mi_lista_desordenada, 42) == 2
14 assert búsqueda_lineal(mi_lista_desordenada, 10) == 0 # Al inicio
15 assert búsqueda_lineal(mi_lista_desordenada, 25) == 6 # Al final
16 assert búsqueda_lineal(mi_lista_desordenada, 99) == -1 # No existe
17 assert búsqueda_lineal([], 5) == -1
18 print("¡Pruebas para búsqueda_lineal pasaron!")
19
20 # 2. Definir la función búsqueda_binaria
21 #definición de función
22 def búsqueda_binaria(lista, elemento_buscado):
23     izquierda = 0
24     derecha = len(lista) - 1
25     while izquierda <= derecha:
26         medio = (izquierda + derecha) // 2
27         if lista[medio] == elemento_buscado:
28             return medio
```

The console on the right shows the output of running the script:

```
~/workspace$ python ejercicios_clase06.py
Probando búsqueda_lineal...
¡Pruebas para búsqueda_lineal pasaron!

Probando búsqueda_binaria...
¡Pruebas para búsqueda_binaria pasaron!

Realizando el experimento del caos...
Resultado del experimento del caos: 1
~/workspace$
```

Ilustración 11 búsqueda lineal, búsqueda binaria y experimento del caos