

# UNIVERSIDAD PRIVADA DOMINGO SAVIO

## FACULTAD DE INGENIERIA



### **Actividad**

**Título: Ejercicios de Replit**

**Docente:** Ing.Jimmy Nataniel Requena Llorentty

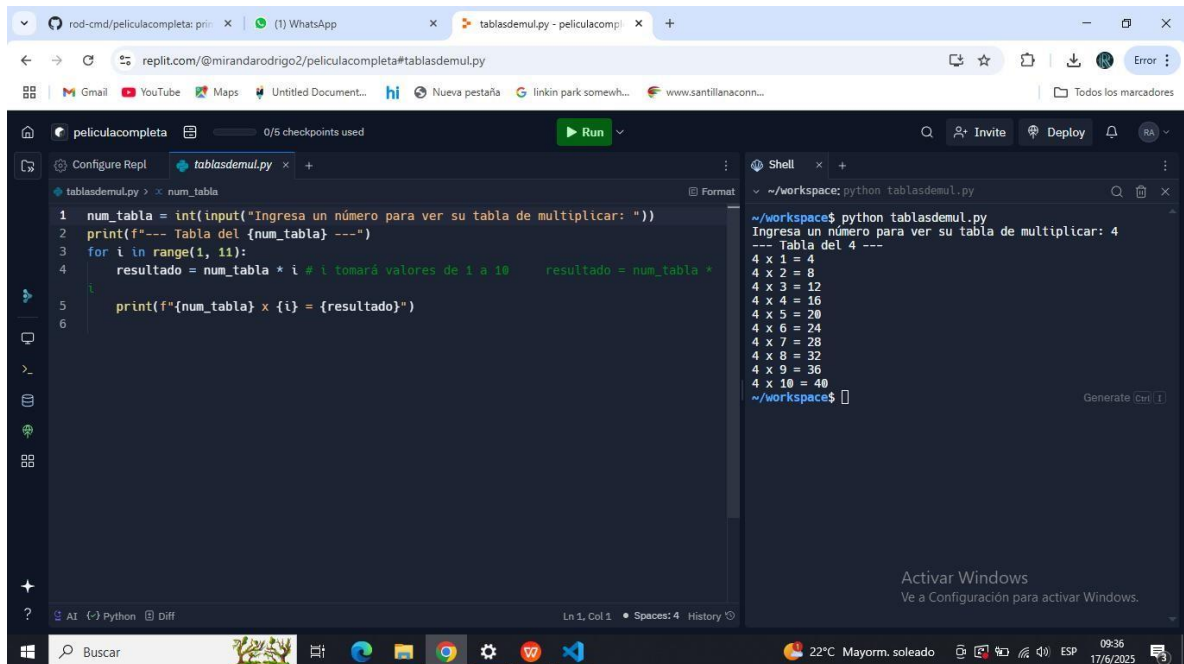
**Materia:** Programación 2

**Estudiante:** Rodrigo Andres Miranda Borda

**Junio del 2025**  
Santa Cruz – Bolivia

## 1. EJERCICIO DE TABLAS DE MULTIPLICACION. \_

El programa pide al usuario ingresar el numero que desee ver la tabla de multiplicar  
Y el programa le da la tabla de multiplicar de dicho número en longitud de 10.



The screenshot shows a Replit workspace with a Python file named `tablasdemul.py`. The code prompts the user to enter a number to see its multiplication table. The user has entered 4, and the program has output the multiplication table for 4, ranging from 4 x 1 to 4 x 10.

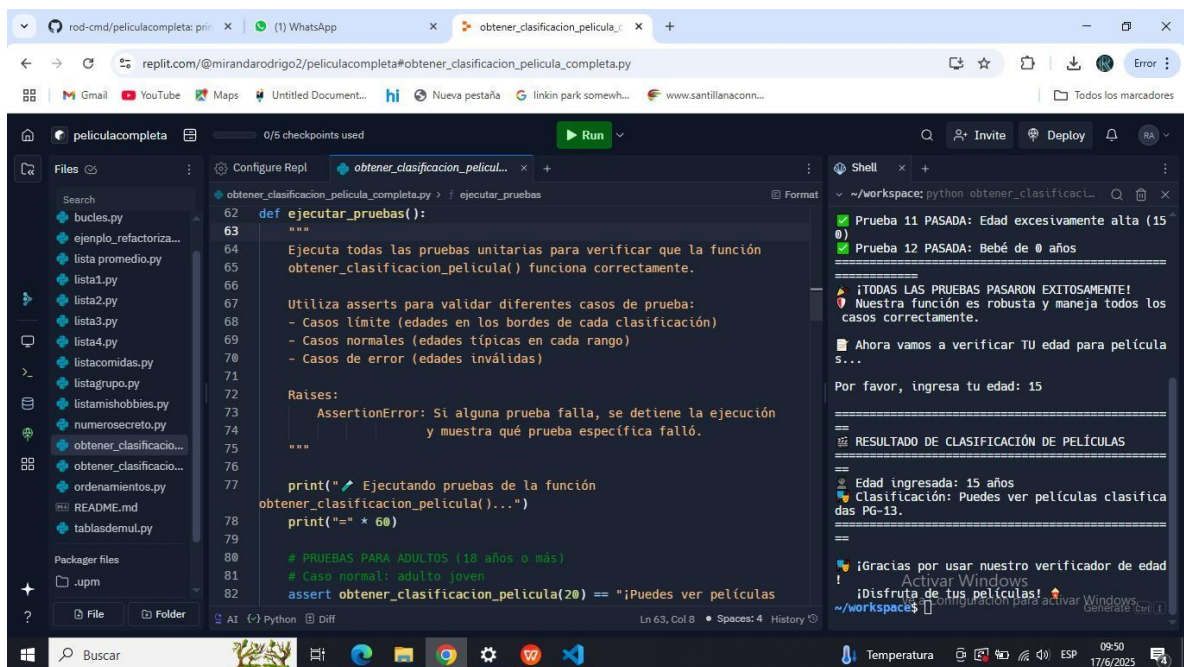
```
1 num_tabla = int(input("Ingresa un número para ver su tabla de multiplicar: "))
2 print(f"--- Tabla del {num_tabla} ---")
3 for i in range(1, 11):
4     resultado = num_tabla * i # i tomará valores de 1 a 10     resultado = num_tabla *
5     print(f"{num_tabla} x {i} = {resultado}")
6
```

The terminal output shows the following:

```
~/workspace$ python tablasdemul.py
Ingresa un número para ver su tabla de multiplicar: 4
--- Tabla del 4 ---
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
~/workspace$
```

## 2. PROGRAMA DE VERIFICACION DE EDAD PARA VER PELICULAS (COMPLETA).

Programa que según tu edad te dice que películas puedes ver versión más completa.



The screenshot shows a Replit workspace with a Python file named `obtener_clasificacion_pelicula.py`. The code defines a function `ejecutar_pruebas()` that runs unit tests for the `obtener_clasificacion_pelicula()` function. The tests include edge cases, normal cases, and error cases. The program also prompts the user to enter their age to see what movies they can watch.

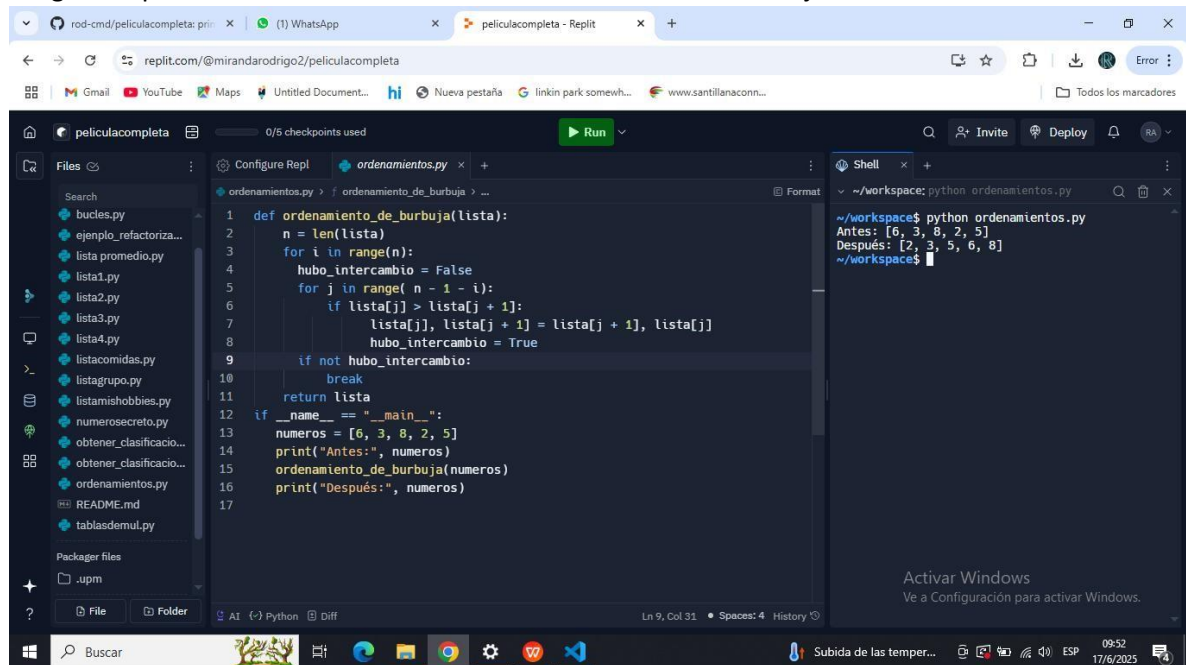
```
62 def ejecutar_pruebas():
63     """
64     Ejecuta todas las pruebas unitarias para verificar que la función
65     obtener_clasificacion_pelicula() funciona correctamente.
66
67     Utiliza asserts para validar diferentes casos de prueba:
68     - Casos límite (edades en los bordes de cada clasificación)
69     - Casos normales (edades típicas en cada rango)
70     - Casos de error (edades inválidas)
71
72     Raises:
73     AssertionError: Si alguna prueba falla, se detiene la ejecución
74     y muestra qué prueba específica falló.
75     """
76
77     print("Ejecutando pruebas de la función
78     obtener_clasificacion_pelicula()...")
79     print("=" * 60)
80
81     # PRUEBAS PARA ADULTOS (18 años o más)
82     # Caso normal: adulto joven
83     assert obtener_clasificacion_pelicula(20) == "¡Puedes ver películas
```

The terminal output shows the following:

```
~/workspace$ python obtener_clasificacion_pelicula.py
Prueba 11 PASADA: Edad excesivamente alta (15 años)
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 película
S...
Por favor, ingresa tu edad: 15
=====
RESULTADO DE CLASIFICACIÓN DE PELÍCULAS
=====
Edad ingresada: 15 años
Clasificación: Puedes ver películas clasificadas PG-13.
=====
¡Gracias por usar nuestro verificador de edad!
¡Disfruta de tus películas!
~/workspace$
```

### 3. LISTA BURBUJA.

Programa que ordena una lista de 5 números con el método burbuja.



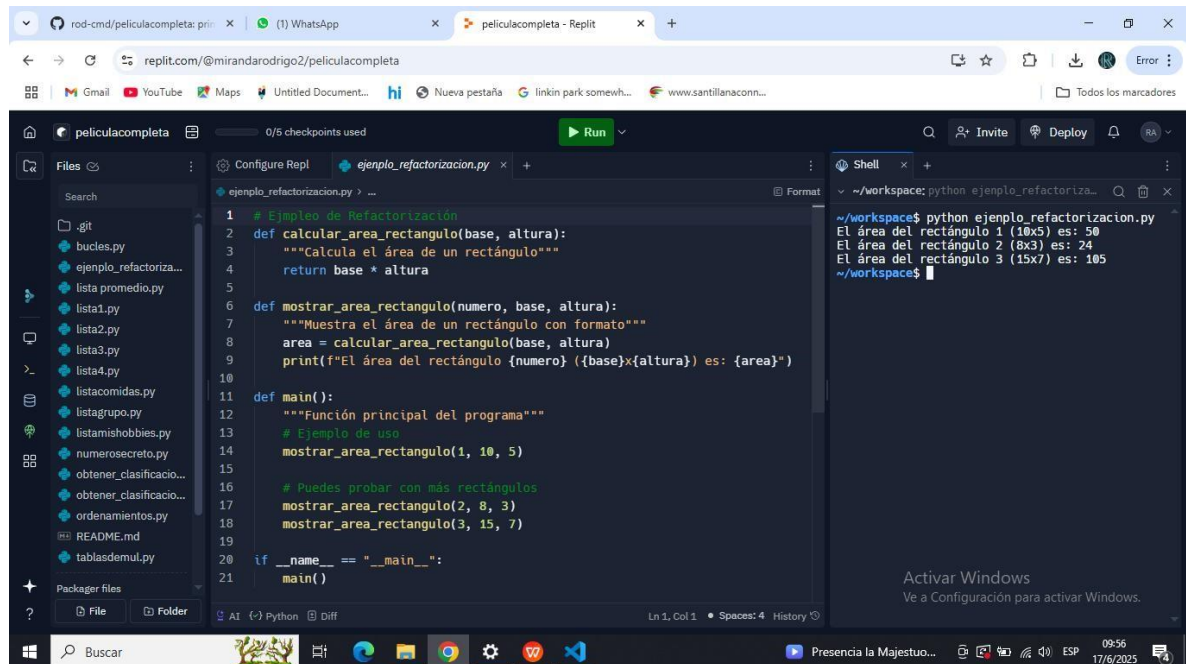
The screenshot shows a Replit workspace named 'peliculacompleta'. The file explorer on the left lists several Python files, including 'ordenamientos.py'. The main editor displays the code for 'ordenamiento\_de\_burbuja.py'. The code defines a function 'ordenamiento\_de\_burbuja' that sorts a list in ascending order using the bubble sort algorithm. It includes a 'main' block with a list of numbers [6, 3, 8, 2, 5] and prints the list before and after sorting. The terminal on the right shows the execution of 'python ordenamientos.py', displaying the output: 'Antes: [6, 3, 8, 2, 5]' and 'Después: [2, 3, 5, 6, 8]'. The Windows taskbar at the bottom shows the time as 09:52 on 17/6/2025.

```
def ordenamiento_de_burbuja(lista):
    n = len(lista)
    for i in range(n):
        hubo_intercambio = False
        for j in range(n - 1 - i):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_intercambio = True
        if not hubo_intercambio:
            break
    return lista

if __name__ == "__main__":
    numeros = [6, 3, 8, 2, 5]
    print("Antes:", numeros)
    ordenamiento_de_burbuja(numeros)
    print("Después:", numeros)
```

### 4. REFACTORIZACION.

Programa que calcula el area de 3 triangulos.



The screenshot shows a Replit workspace named 'peliculacompleta'. The file explorer on the left lists several Python files, including 'ejemplo\_refactorizacion.py'. The main editor displays the code for 'ejemplo\_refactorizacion.py'. The code defines a function 'calcular\_area\_rectangulo' that calculates the area of a rectangle given its base and height. It also defines a function 'mostrar\_area\_rectangulo' that prints the area with formatted output. The 'main' block uses these functions to calculate and display the areas of three rectangles. The terminal on the right shows the execution of 'python ejemplo\_refactorizacion.py', displaying the output: 'El área del rectángulo 1 (10x5) es: 50', 'El área del rectángulo 2 (8x3) es: 24', and 'El área del rectángulo 3 (15x7) es: 105'. The Windows taskbar at the bottom shows the time as 09:56 on 17/6/2025.

```
# Ejemplo de Refactorización
def calcular_area_rectangulo(base, altura):
    """Calcula el área de un rectángulo"""
    return base * altura

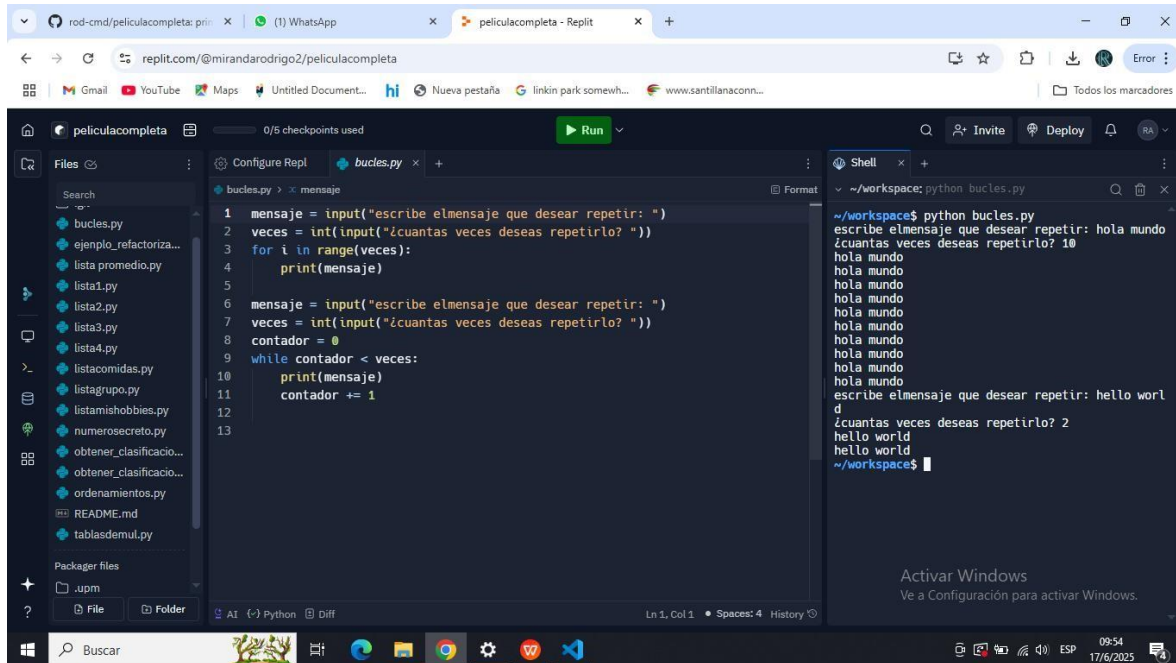
def mostrar_area_rectangulo(numero, base, altura):
    """Muestra el área de un rectángulo con formato"""
    area = calcular_area_rectangulo(base, altura)
    print(f"El área del rectángulo {numero} ({base}x{altura}) es: {area}")

def main():
    """Función principal del programa"""
    # Ejemplo de uso
    mostrar_area_rectangulo(1, 10, 5)

    # Puedes probar con más rectángulos
    mostrar_area_rectangulo(2, 8, 3)
    mostrar_area_rectangulo(3, 15, 7)

if __name__ == "__main__":
    main()
```

**5. BUCLES.** Ejercicios de bucles 1 con for y unos con while, donde ambos programas piden al usuario Ingresar un texto y pedir cuantas veces sea mostrado ese texto.



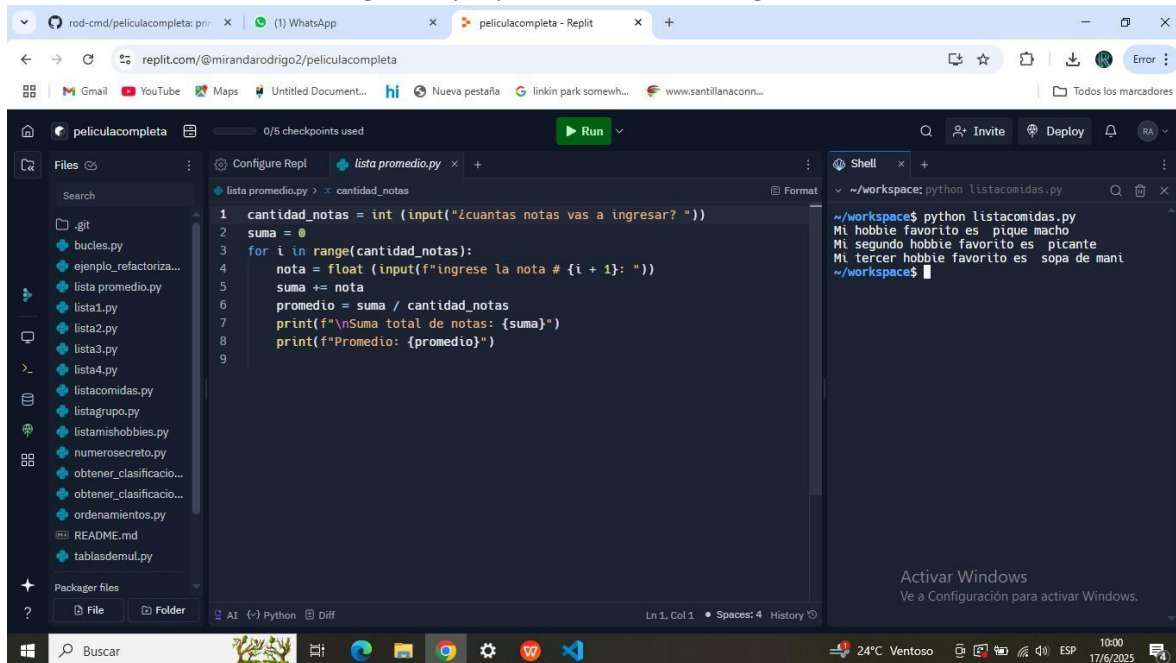
The screenshot shows a Replit environment with a file explorer on the left containing various Python files. The main editor displays a file named `bucles.py` with the following code:

```
1 mensaje = input("escribe el mensaje que deseas repetir: ")
2 veces = int(input("¿cuántas veces deseas repetirlo? "))
3 for i in range(veces):
4     print(mensaje)
5
6 mensaje = input("escribe el mensaje que deseas repetir: ")
7 veces = int(input("¿cuántas veces deseas repetirlo? "))
8 contador = 0
9 while contador < veces:
10    print(mensaje)
11    contador += 1
12
13
```

The right-hand pane shows the terminal output of the script:

```
~/workspace$ python bucles.py
escribe el mensaje que deseas repetir: hola mundo
¿cuántas veces deseas repetirlo? 10
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
escribe el mensaje que deseas repetir: hello world
¿cuántas veces deseas repetirlo? 2
hello world
hello world
~/workspace$
```

**6 . PROMEDIO DE NOTAS.** Programa que promedia las notas ingresadas.



The screenshot shows a Replit environment with a file explorer on the left. The main editor displays a file named `lista_promedio.py` with the following code:

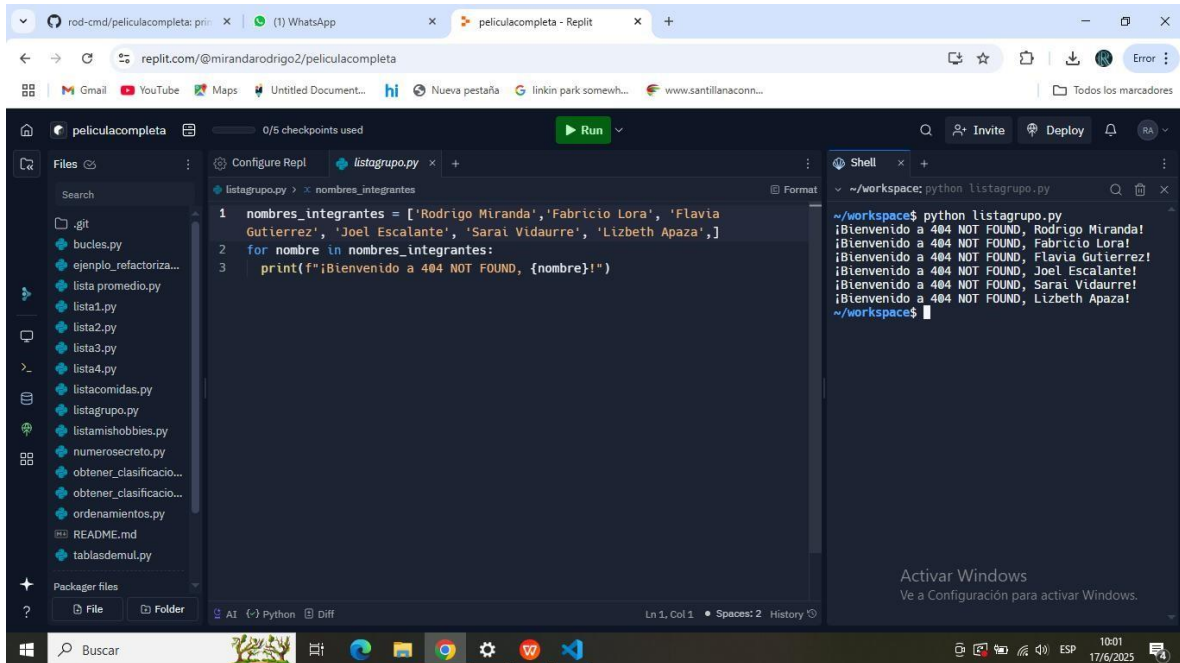
```
1 cantidad_notas = int (input("¿cuántas notas vas a ingresar? "))
2 suma = 0
3 for i in range(cantidad_notas):
4     nota = float (input(f"ingrese la nota # {i + 1}: "))
5     suma += nota
6 promedio = suma / cantidad_notas
7 print(f"\nSuma total de notas: {suma}")
8 print(f"Promedio: {promedio}")
9
```

The right-hand pane shows the terminal output of the script:

```
~/workspace$ python listacomidas.py
Mi hobbie favorito es pique macho
Mi segundo hobbie favorito es picante
Mi tercer hobbie favorito es sopa de mani
~/workspace$
```



## 7. LISTA DE LOS INTEGRANTES DE MI GRIPO. Programa que me lista los integrantes de mi grupo dandoles la bienvenida al grupo.

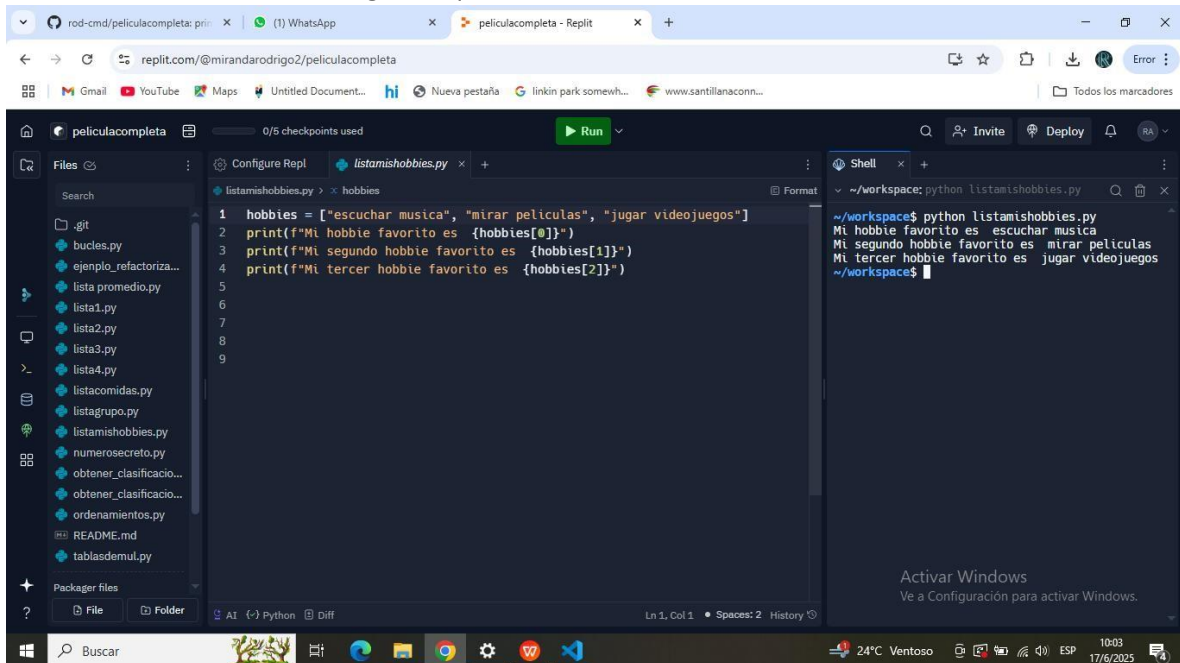


The screenshot shows a Replit workspace for a project named 'peliculacompleta'. The file explorer on the left lists various Python files, including 'listagrupo.py'. The main editor displays the code for 'listagrupo.py', which defines a list of names and prints a welcome message for each. The shell on the right shows the command 'python listagrupo.py' being executed, resulting in five lines of welcome messages for the group members.

```
1 nombres_integrantes = ['Rodrigo Miranda', 'Fabricio Lora', 'Flavia Gutierrez', 'Joel Escalante', 'Sara Vidaurre', 'Lizbeth Apaza',]  
2 for nombre in nombres_integrantes:  
3     print(f"¡Bienvenido a 404 NOT FOUND, {nombre}!")
```

```
~/workspace$ python listagrupo.py  
¡Bienvenido a 404 NOT FOUND, Rodrigo Miranda!  
¡Bienvenido a 404 NOT FOUND, Fabricio Lora!  
¡Bienvenido a 404 NOT FOUND, Flavia Gutierrez!  
¡Bienvenido a 404 NOT FOUND, Joel Escalante!  
¡Bienvenido a 404 NOT FOUND, Sara Vidaurre!  
¡Bienvenido a 404 NOT FOUND, Lizbeth Apaza!  
~/workspace$
```

## 8. LISTA DE MIS HOBBIES. Programa que me lista mis 3 hobbies favoritos.



The screenshot shows a Replit workspace for the same project. The file explorer lists 'listamishobbies.py'. The main editor displays the code for 'listamishobbies.py', which defines a list of hobbies and prints them out. The shell on the right shows the command 'python listamishobbies.py' being executed, resulting in three lines of hobby names.

```
1 hobbies = ["escuchar musica", "mirar peliculas", "jugar videojuegos"]  
2 print(f"Mi hobbie favorito es {hobbies[0]}")  
3 print(f"Mi segundo hobbie favorito es {hobbies[1]}")  
4 print(f"Mi tercer hobbie favorito es {hobbies[2]}")  
5  
6  
7  
8  
9
```

```
~/workspace$ python listamishobbies.py  
Mi hobbie favorito es escuchar musica  
Mi segundo hobbie favorito es mirar peliculas  
Mi tercer hobbie favorito es jugar videojuegos  
~/workspace$
```

**9. NUMERO SECRETO.** Programa para adivinar un numero secreto, donde al usuario se le da pistas en cada turno si el nro es muy alto o muy bajo en un rango de 10 numeros.

```
1 numero_secreto = 7
2 intento = int(input("Adivina el número secreto (entre 1 y 10): "))
3 while intento != numero_secreto:
4     if intento > numero_secreto:
5         print("Demasiado alto.")
6     else:
7         print("Demasiado bajo.")
8     intento = int(input("Intenta otra vez: "))
9 print(f"¡Correcto! El número era {numero_secreto}.")
```

```
~/workspace$ python numero_secreto.py
Adivina el número secreto (entre 1 y 10): 4
Demasiado bajo.
Intenta otra vez: 6
Demasiado bajo.
Intenta otra vez: 9
Demasiado alto.
Intenta otra vez: 7
¡Correcto! El número era 7.
~/workspace$
```

**10. LISTA EL ELEMENTO.** Programa que me dice cuantas veces se repite un un elemento numerico.

```
1 def contar_ocurrencias(lista, elemento_buscado):
2     contador = 0 # Inicializa el contador en 0
3     for elemento in lista: # Itera sobre cada elemento de la lista
4         if elemento == elemento_buscado: # Si el elemento es igual al
5             elemento_buscado
6             contador += 1 # Incrementa el contador en 1
7     return contador # Retorna el contador # Fin de la función
8
9 mis_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
10 elemento_buscado = 10
11 resultado = contar_ocurrencias(mis_numeros, elemento_buscado)
12 print(f"El elemento {elemento_buscado} aparece {resultado} veces en la
13 lista.")
```

```
~/workspace$ python lista1.py
El elemento 10 aparece 1 veces en la lista.
~/workspace$
```

## 11. LISTA INVERTIDA. Programa invierte los elementos de una lista.

```
1 lista_invertida = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
2 print('Lista original:', lista_invertida)
3 lista_invertida.reverse()
4 print('Lista invertida:', lista_invertida)
5
6
```

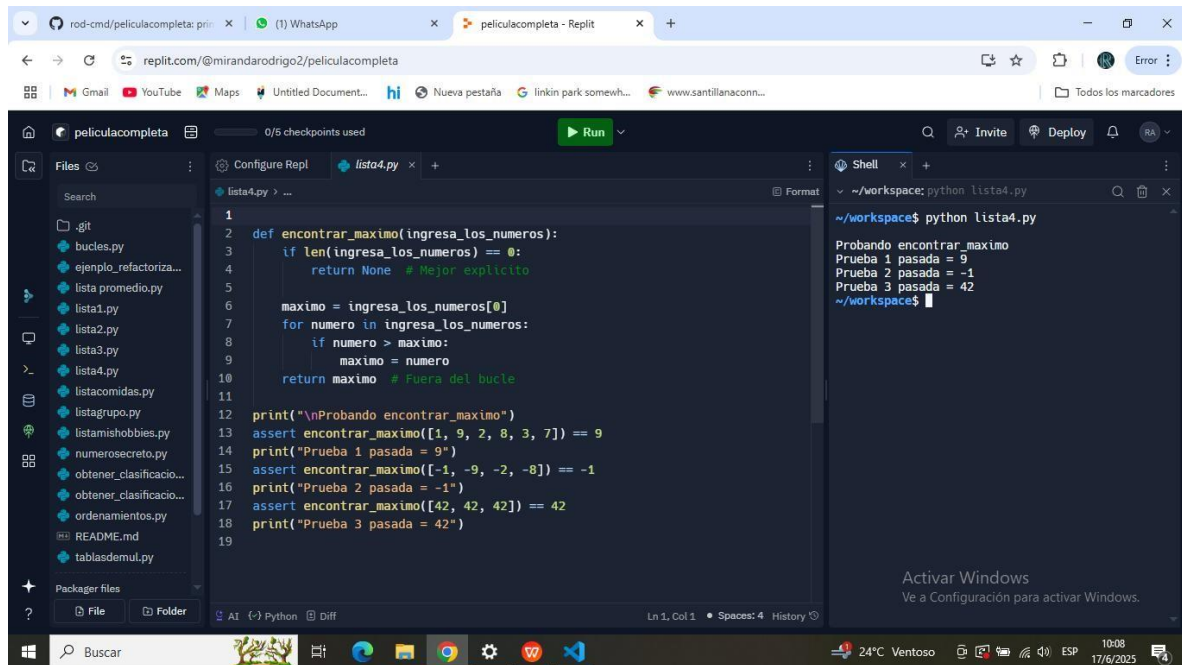
```
~/workspace$ python lista2.py
Lista original: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
Lista invertida: [5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

## 12. SUMA LISTA DE ELEMENTOS. Programa que suma una lista de elementos, distintas pruebas con assert.

```
1 def sumar_elementos(lista_numeros):
2     acumulador_suma = 0
3     for numero in lista_numeros:
4         acumulador_suma += numero
5     return acumulador_suma
6
7
8 mi_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
9 resultado = sumar_elementos(mi_lista)
10
11 print(f'La suma de los elementos de la lista es: {resultado}')
12
13 print('-Con assert-')
14
15 def sumar_elementos(lista_numeros):
16     acumulador_suma = 0
17     for numero in lista_numeros:
18         acumulador_suma += numero
19     return acumulador_suma
20 assert sumar_elementos([1, 2, 3, 4, 5]) == 15
21 print('Prueba 1 pasada = 15')
22 assert sumar_elementos([10, 20, 30, 40, 50]) == 150
23
```

```
~/workspace$ python lista3.py
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.
```

### 13. ENCONTRAR NUMERO MAYOR. Programa que encuentra el numero mayor de una lista, pruebas con assert.

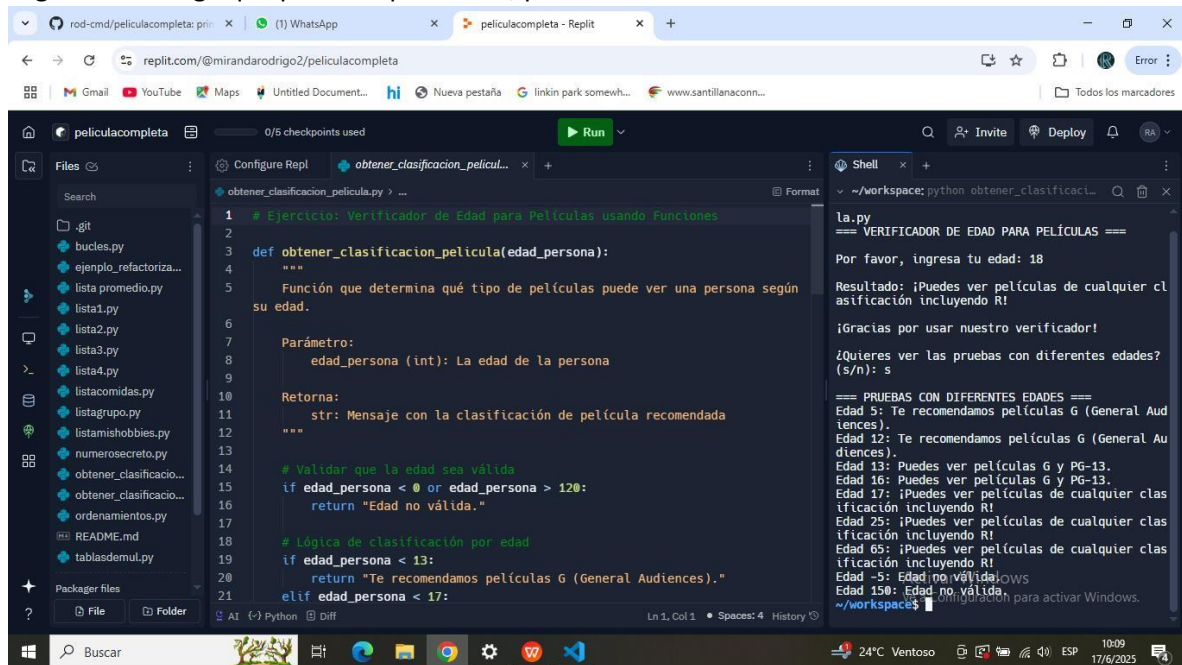


The screenshot shows a Replit workspace for a project named 'peliculacompleta'. The file explorer on the left lists several Python files, including 'lista4.py'. The main editor displays the code for 'lista4.py', which defines a function 'encontrar\_maximo' to find the maximum value in a list. The function includes a base case for an empty list and a loop to iterate through the list. Test cases are included using the 'assert' statement to verify the function's output for specific inputs. The shell on the right shows the execution of the script, displaying the results of the tests.

```
1 def encontrar_maximo(ingresa_los_numeros):
2     if len(ingresa_los_numeros) == 0:
3         return None # Mejor explicito
4
5     maximo = ingresa_los_numeros[0]
6     for numero in ingresa_los_numeros:
7         if numero > maximo:
8             maximo = numero
9     return maximo # Fuera del bucle
10
11
12 print("\nProbando encontrar_maximo")
13 assert encontrar_maximo([1, 9, 2, 8, 3, 7]) == 9
14 print("Prueba 1 pasada = 9")
15 assert encontrar_maximo([-1, -9, -2, -8]) == -1
16 print("Prueba 2 pasada = -1")
17 assert encontrar_maximo([42, 42, 42]) == 42
18 print("Prueba 3 pasada = 42")
19
```

```
~/workspace$ python lista4.py
Probando encontrar_maximo
Prueba 1 pasada = 9
Prueba 2 pasada = -1
Prueba 3 pasada = 42
~/workspace$
```

### 14. VERIFICADOR DE EDAD PARA VER PELICULAS. Programa para verificar la edad del usuario y segun esta le diga que peliculas puede ver, primera version.



The screenshot shows a Replit workspace for a project named 'peliculacompleta'. The file explorer on the left lists several Python files, including 'obtener\_clasificacion\_pelicula.py'. The main editor displays the code for 'obtener\_clasificacion\_pelicula.py', which defines a function 'obtener\_clasificacion\_pelicula' to determine the movie rating based on the user's age. The function includes a base case for an empty list and a loop to iterate through the list. Test cases are included using the 'assert' statement to verify the function's output for specific inputs. The shell on the right shows the execution of the script, displaying the results of the tests.

```
1 # Ejercicio: Verificador de Edad para Peliculas usando Funciones
2
3 def obtener_clasificacion_pelicula(edad_persona):
4     """
5     Función que determina qué tipo de películas puede ver una persona según
6     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     # Validar que la edad sea válida
15     if edad_persona < 0 or edad_persona > 120:
16         return "Edad no válida."
17
18     # Lógica de clasificación por edad
19     if edad_persona < 13:
20         return "Te recomendamos películas G (General Audiences)."
21     elif edad_persona < 17:
```

```
la.py
=== VERIFICADOR DE EDAD PARA PELICULAS ===

Por favor, ingresa tu edad: 18

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 Audiences).
Edad 12: Te recomendamos películas G (General Audiences).
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.
~/workspace$
```



