**UNIVERSIDAD PRIVADA DOMINGO SAVIO**



**Ejercicios de programación II: Actividad 01**

**Docente:**

Jimmy Nataniel Requena Llorentty

**Estudiante:**

Rebeca Vargas Orellana

**Materia:**

Programación II – Turno mañana

*Santa Cruz – Bolivia*

Ejercicios de Programación

Actividad 01: Clases de Programación II

|  |
| --- |
| edad\_str = input("Bienvenido al cine, ¿Cual es tu edad?:" )  edad = int(edad\_str)  if edad < 0:  print ("Edad no valida. por favor, ingresa un numero positivo.")  elif edad >= 18:  print ("Puedes ver peliculas clasificadas R!")  elif edad >= 13: # la edad no es >= 18 y < 0  print ("Puedes ver peliculas clasificadas PG-13.")  elif edad < 13:  print ("Puedes ver peliculas clasificadas A.") |
|  |
| Código: Cine |

|  |
| --- |
| Edad\_str = input("Bienvenido,por favor ingresa tu edad:")  Edad= int(Edad\_str)  if Edad >= 18: #Condicion 1  print("Es mayor de edad. ¡Puede votar y conducir (con licencia)")  elif Edad >= 13: #Condicion 2  print ("Es un adolescente.")  elif Edad < 0: #Condicion 3  print("Edad invalida. ¡Los viajes en el tiempo no son comerciales!")  else: #Bloque opcional  print("es un niño o niña.") |
|  |
| Código: Definición Edad |

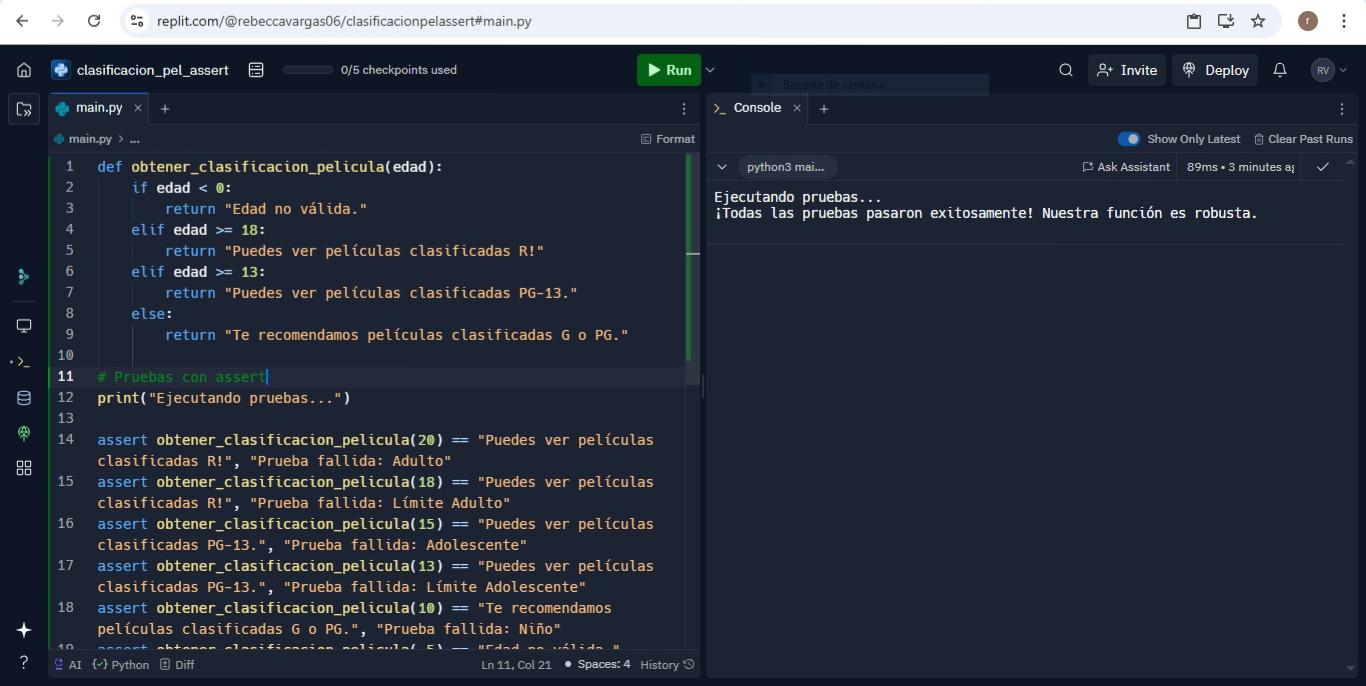
|  |  |
| --- | --- |
|  |  |
| Código: Tabla de Multiplicar |  |

|  |
| --- |
| # Bucle for  print("Contando hasta 3 (sin incluirlo):")  for numero in range(3): # range(3) genera 0, 1, 2  print(numero)  print("\nRecorriendo un string:")  nombre = "PYTHON"  for letra in nombre:  print(letra) |
|  |
| Código: Bucle For |

|  |
| --- |
| # Bucle while  contador = 0  print("\nBucle while:")  while contador < 3:  print(f"Contador es: {contador}")  contador = contador + 1  print("¡Bucle while terminado!") |
|  |
| Código: Bucle While |

|  |  |
| --- | --- |
|  |  |
| Código: Saludo |  |

|  |
| --- |
| def obtener\_clasificacion\_pelicula(edad):  if edad < 0:  return "Edad no válida."  elif edad >= 18:  return "Puedes ver películas clasificadas R!"  elif edad >= 13:  return "Puedes ver películas clasificadas PG-13."  else:  return "Te recomendamos películas clasificadas G o PG."  # Pruebas con assert  print("Ejecutando pruebas...")  assert obtener\_clasificacion\_pelicula(20) == "Puedes ver películas clasificadas R!", "Prueba fallida: Adulto"  assert obtener\_clasificacion\_pelicula(18) == "Puedes ver películas clasificadas R!", "Prueba fallida: Límite Adulto"  assert obtener\_clasificacion\_pelicula(15) == "Puedes ver películas clasificadas PG-13.", "Prueba fallida: Adolescente"  assert obtener\_clasificacion\_pelicula(13) == "Puedes ver películas clasificadas PG-13.", "Prueba fallida: Límite Adolescente"  assert obtener\_clasificacion\_pelicula(10) == "Te recomendamos películas clasificadas G o PG.", "Prueba fallida: Niño"  assert obtener\_clasificacion\_pelicula(-5) == "Edad no válida.", "Prueba fallida: Edad negativa"  print("¡Todas las pruebas pasaron exitosamente! Nuestra función es robusta. ")  Código: Películas Assert |
|  |



|  |  |
| --- | --- |
|  |  |
| Código: Clasificación películas Assert |  |

|  |
| --- |
| Def invertir\_lista(lista\_original):  # Crear una nueva lista vacía  Lista\_invertida = []  # Recorrer la lista original desde el final hacia el inicio  For i in range(len(lista\_original) – 1, -1, -1):  Lista\_invertida.append(lista\_original[i])  # Retornar la nueva lista invertida  Return lista\_invertida  # Pruebas  Print(“\nProbando invertir\_lista…”)  Lista\_prueba = [1, 2, 3, 4, 5]  Lista\_resultante = invertir\_lista(lista\_prueba)  Assert lista\_resultante == [5, 4, 3, 2, 1]  Assert lista\_prueba == [1, 2, 3, 4, 5] # Verifica que la original no cambió  Assert invertir\_lista([“a”, “b”, “c”]) == [“c”, “b”, “a”]  Assert invertir\_lista([]) == []  Print(“¡Pruebas para invertir\_lista pasaron! ✅”) |
|  |
| Código: Lista Invertida |

|  |
| --- |
| def encontrar\_mayor(lista\_numeros):  # Caso especial: lista vacía  if not lista\_numeros:  return None  # Asignar el primer elemento como el mayor temporal  mayor\_temporal = lista\_numeros[0]  # Recorrer el resto de la lista  for elemento\_actual in lista\_numeros[1:]:  if elemento\_actual > mayor\_temporal:  mayor\_temporal = elemento\_actual  return mayor\_temporal  # Pruebas  print("\nProbando encontrar\_mayor...")  assert encontrar\_mayor([1, 9, 2, 8, 3, 7]) == 9  assert encontrar\_mayor([-1, -9, -2, -8]) == -1  assert encontrar\_mayor([42, 42, 42]) == 42  assert encontrar\_mayor([]) == None # Prueba del caso especial  assert encontrar\_mayor([5]) == 5  print("¡Pruebas para encontrar\_mayor pasaron! ") |
|  |
| Código: Encontrar número mayor |

|  |
| --- |
| def ordenamiento\_burbuja(lista):  n = len(lista)  for i in range (n - 1):  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\_burbuja(numeros)  print("Despues:", numeros) |
|  |
| Código: Ordenamiento burbuja |

|  |
| --- |
| # Definición de la función de ordenamiento burbuja  def ordenamiento\_burbuja(lista):  n = len(lista)  for i in range(n):  for j in range(0, n - i - 1):  if lista[j] > lista[j + 1]:  lista[j], lista[j + 1] = lista[j + 1], lista[j]  #llamar a la función:  lista\_a\_ordenar = [64, 34, 25, 12, 22, 11, 90]  print(f"Lista original: {lista\_a\_ordenar}")  ordenamiento\_burbuja(lista\_a\_ordenar) # Llamamos a la función  print(f"Lista ordenada: {lista\_a\_ordenar}")  # Prueba con assert:  # Caso 1: Lista desordenada  lista1 = [6, 3, 8, 2, 5]  ordenamiento\_burbuja(lista1)  assert lista1 == [2, 3, 5, 6, 8], "Falló en Caso 1"  # Caso 2: Lista ya ordenada  lista2 = [1, 2, 3, 4, 5]  ordenamiento\_burbuja(lista2)  assert lista2 == [1, 2, 3, 4, 5], "Falló en Caso 2"  # Caso 3: Lista ordenada a la inversa (peor caso)  lista3 = [5, 4, 3, 2, 1]  ordenamiento\_burbuja(lista3)  assert lista3 == [1, 2, 3, 4, 5], "Falló en Caso 3"  # Caso 4: Lista con elementos duplicados  lista4 = [5, 1, 4, 2, 5, 5, 2]  ordenamiento\_burbuja(lista4)  assert lista4 == [1, 2, 2, 4, 5, 5, 5], "Falló en Caso 4"  # Casos borde  assert ordenamiento\_burbuja([]) == None  assert ordenamiento\_burbuja([42]) == None, "Fallo en caso borde"  print("¡Todas las pruebas pasaron!") |
|  |
| Código: Ordenamiento de menor a mayor |

+