



**Instituto Tecnológico José Mario Molina Pasquel y
Henríquez**

Unidad Académica Zapotlanejo.

Carrera: Ingeniería Informática (semestre 3)

Materia: Estructura de datos

Actividad: Manual de practicas

Autor: Cristian Rodriguez Rodriguez

Parcial: III

Zapotlanejo, Jalisco, 07 Noviembre de 2025

Índice

Practica 1 (Sintaxis)	4
	5
Practica 2 (Uso de listas)	6
Practica 3 (Inicializar arreglos)	7
Practica 4(Sumar datos de una lista)	9
Tarea 1(Guardar en arreglo y lista)	11
Practica 5(resolución Tarea 1).....	12
Practica 6(validación simple)	14
Practica 7(Guardar registros)	15
Practica 8 (Contador Letras)	17
Tarea 2 (Identificar vocales, espacios, mayúsculas)	19
Practica 9 (Resolución Tarea 2).....	20
Repasso 1(Calculo de costos)	23
Repasso 2 (Formula general).....	24
Repasso 3 (Clasificaciones).....	25
Repasso 3 (Resolución clase).....	26
Unidad 2	27
Practica 1	27
Practica 2	28
Validaciones Practica 2	29
Practica 3:	29
Practica 4:	32
Practica 5	34
Practica 6	37
Practica 7	40
Repasso 1	42
Actividad Clase (Resolución Examen Parcial 1):	45
Tarea 1 (Solución de examen Parcial 1)	48

Repaso 2 (preexamen).....	51
Parcial 3: 54	
Practica 1 (Ordenamiento):	54
Practica 2 (Actualización de la practica 1 / Place-Holder):	57
Practica 3:	59
Practica 4(Tabla):	63
Repaso 1(Tarea 1: Generar RFC):	66
Validaciones:	69
Unidad 4 71	
Practica 1 (Suma de Números)	71
Practica 2(Registro de usuarios):	74
Practica 3 (Sistema de Tienda):	80
Tarea 1 (Mostrar numero en listBox):	86

Practica 1 (Sintaxis)

En este programa se analizan la sintaxis básica para realizar operaciones aritméticas en Python

```
# a = 0
# variable reservada (comentada)

# print("escribe un numero")
# mensaje de instrucción (comentado)

a = int(input('escribe un numero: '))
# lee un número del usuario y lo convierte a entero
# reads a number from user and converts it to integer

print(a ** 2) # calcularía el cuadrado (ej: 5 + 25)
# would calculate square (e.g., 5 + 25)

print(a ** (1/2)) # calcula la raíz cuadrada (ej: 25 → 5.0)
# calculates square root (e.g., 25 → 5.0)

# +
# suma: suma dos números → 3 + 2 = 5
# addition: adds two numbers → 3 + 2 = 5

# -
# resta: resta dos números → 5 - 2 = 3
# subtraction: subtracts two numbers → 5 - 2 = 3

# *
# multiplicación: multiplica dos números → 4 * 3 = 12
# multiplication: multiplies two numbers → 4 * 3 = 12

# /
# división exacta: devuelve resultado con decimales → 7 / 2 = 3.5
# exact division: returns result with decimals → 7 / 2 = 3.5

# //
# división entera: devuelve solo la parte entera (trunca) → 7 // 2 = 3
# floor division: returns only integer part (truncates) → 7 // 2 = 3

# **
# potencia: eleva un número a otro → 2 ** 3 = 8
# exponentiation: raises a number to a power → 2 ** 3 = 8

# mod → en Python es %: devuelve el residuo → 7 % 3 = 1
# mod → in Python it's %: returns remainder → 7 % 3 = 1

# and
# operador lógico: verdadero solo si AMBOS son verdaderos
# logical operator: true only if BOTH are true
# ej: (5 > 3) and (2 < 4) → True

# or
# operador lógico: verdadero si AL MENOS UNO es verdadero
# logical operator: true if AT LEAST ONE is true
# ej: (5 < 3) or (2 < 4) → True
```

```
# ej: (5 < 3) or (2 < 4) → True

# <, >, >=, !=, ==, not
# operadores de comparación
# comparison operators
# < : menor que
# > : mayor que
# >= : mayor o igual
# <= : menor o igual
# != : diferente
# == : igual (compara valor)
# not: invierte un valor booleano

...
~~~comentario de varias líneas
multi-line comment
...
~~~
```

Ilustración 1 Código práctica 1

```
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> & C:/  
EMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa1.py"  
escribe un numero: 12  
3.4641016151377544  
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> █
```

Ilustración 2 Consola Práctica 1

Practica 2 (Uso de listas)

Descripción: En esta practica se trabaja con listas, aprendimos a agregar datos nuevos al final de la misma, y a inicializarlas.

```

1  a = [10, 23, 14, 15] # arreglo: convención de un solo tipo (números), tamaño definido - finito
2      # array: convention of single type (numbers), fixed size - finite
3  b = []                 # lista: puede crecer y contener varios tipos - dinámica e "infinita"
4      # list: can grow and hold multiple types - dynamic and "infinite"
5
6  a[0] = 10               # cambia el primer valor de 'a' a 10 (aunque ya era 10)
7      # changes first value of 'a' to 10 (even though it was already 10)
8
9  b.append(12)            # agrega el valor 12 al final de la lista 'b'
10     # adds the value 12 to the end of list 'b'
11
12 # b = {"hola", 10, 10.05, False, 'm', {1, 2, 3, 4}}1
13 # línea comentada: esto no es una lista, es un 'set' (conjunto), y no permite listas dentro
14
15 # ciclos y comparación de tamaño
16 # loops and size comparison
17 if (len(a) > len(b)):
18     print("A es mayor")
19 else:2
20     print("B es mayor")
21
22 # repite 3 veces: i toma valores 0, 1, 2
23 # repeats 3 times: i takes values 0, 1, 2
24 for i in range(3):
25     print(i)
26
27 # recorre cada elemento en 'a'
28 # iterates through each element in 'a'
29 for i in a:
30     print(a) # imprime toda la lista 'a' en cada vuelta → [10,23,14,15] repetido 4 veces
31     # prints entire list 'a' each time → [10,23,14,15] repeated 4 times

```

Ilustración 3 Código Practica 2

```

A es mayor
0
1
2
[10, 23, 14, 15]
[10, 23, 14, 15]
[10, 23, 14, 15]
[10, 23, 14, 15]
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> []

```

Ilustración 4 Consola practica 2

Practica 3 (Inicializar arreglos)

Descripción: El programa solicita 10 números y los almacena en un arreglo, ya inicializado, usando un bucle for, que opera en un rango de 0 a 10.

```
1  # hacer un programa que lea 10 numeros y los almacene en un arreglo
2  # make a program that reads 10 numbers and stores them in an array
3  num = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4
5  # 1: arreglo de 10 posiciones, inicializado en cero
6  # 1: array of 10 positions, initialized to zero
7  # 2: se llenará con los números ingresados por el usuario
8  # 2: will be filled with numbers entered by the user
9
10 # bucle que recorre dentro de un rango de 0 a 9 (10 posiciones)
11 # loop that iterates over the range 0 to 9 (10 positions)
12 for i in range(0, 10):
13     num[i] = int(input(f"Escribe un numero {i + 1}: \n "))
14
15 # imprime cada número almacenado en el arreglo
16 # prints each number stored in the array
17 for i in num:
18     print(i)
```

Ilustración 5 Código Practica 3

```
xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa3.py"
Escribe un numero 1:
23
Escribe un numero 2:
34
Escribe un numero 3:
342
Escribe un numero 4:
432
Escribe un numero 5:
43
Escribe un numero 6:
443
Escribe un numero 7:
12
Escribe un numero 8:
445
Escribe un numero 9:
32
Escribe un numero 10:
34
23
34
342
432
43
443
12
445
32
34
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> █
```

Ilustración 6 Consola Practica 3

Practica 4(Sumar datos de una lista)

Descripción: Se piden 10 números, los cuales se almacenan en una lista, y al final de pedir los números, se le regresa al usuario la suma de estos.

```

1  # hacer unprograma que lea 10 numeros y los almacene en un arreglo
2  # make a program that reads 10 numbers and stores them in an array
3
4  a = []
5  s = 0
6  n = 0
7  numeros = "0,1,2,3,4,5,6,7,8,9" # tambien se puede: "123456789";
8  while(n < 10):
9
10     b = input("escribe un numero: ")
11     x = 0
12
13     for i in b: # se recorre los elementos del numero
14         | | # iterates through the elements of the number
15
16         # if(ord(i) >= 48 and ord(i) <= 57): # se pregunta si son numeros
17         if i in numeros: # asks if they are numbers
18             x += 1
19
20         if len(b) == x: # si todos coinciden son numeros y se agregan a la lista
21             a.append(int(b)) # if all the numbers match, they are added to the list
22             n += 1
23         else:
24             print("El valor no es numero")
25
26
27
28     for i in a:
29         print(i)
30         s += i
31     #print(a) # imprime toda la lista, incluyendo los corchetes
32     | | # prints the entire list, including the brackets
33
34     print(f"la suma es:{s}")

```

Ilustración 7 Código Practica 4

```
xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa4.py"
escribe un numero: 1
escribe un numero: 2
escribe un numero: 3
escribe un numero: 4
escribe un numero: 5
escribe un numero: 6
escribe un numero: 7
escribe un numero: 8
escribe un numero: 9
escribe un numero: 0
1
2
3
4
5
6
7
8
9
0
la suma es:45
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> □
```

Ilustración 8 Consola Practica 4

Tarea 1(Guardar en arreglo y lista)

El programa que le 10 datos, si el dato es un numero, se almacenara en un arreglo si es un carácter o caracteres se meterá en una lista, cuando finalice el programa nos mostrara cuantos elementos numéricos y cuantos caracteres hay en cada estructura.

```

2  hacer un programa que lea 10 datos, si el dato es un numero, se almacenara en un arreglo
3  si es un caracter o caracteres se metera en una lista, cuando finalice el programa nos mostrara
4  cuantos elementos numericos y cuantos caracteres hay en cada estructura
5  ...
6
7  # declarar los arreglos y la lista
8  # declare the arrays and the list
9  # num = [0,0,0,0,0,0,0,0,0,0] # arreglo fijo de 10 posiciones para números (inicializado con ceros)
10 num = [0]*10                # fixed array of 10 positions for numbers (initialized with zeros)
11 pos = 0                     # posición actual en el arreglo 'num' (cuántos números llevamos guardados)
12 |                         |   # current position in the 'num' array (how many numbers we have stored)
13 cract = []                  # lista para almacenar caracteres o cadenas de texto
14 |                         |   # list to store characters or text strings
15
16 # preguntar 10 datos
17 # ask for 10 data inputs
18 n = 0                       # contador de datos ingresados (de 0 a 9)
19 |                         |   # counter for entered data (from 0 to 9)
20 ~ while(n < 10):
21     aux = input("datos: ")    # leer entrada del usuario
22     |                         |   # read user input
23     cont = 0                 # variable no utilizada (podría eliminarse, pero no se modifica el código)
24     |                         |   # unused variable (could be removed, but code is not modified)
25
26 # ver si es caracter o numero
27 # check if it is a character or a number
28 ~ if aux.isdigit():
29     num[pos] = int(aux)      # si todos los caracteres son dígitos (ej: "123", válido como número)
30     |                         |   # if all characters are digits (e.g., "123", valid as number)
31     num[pos] = int(aux)      # convertir a entero y guardarla en la posición actual de 'num'
32     |                         |   # convert to integer and store in current 'num' position
33     pos += 1                 # avanzar a la siguiente posición en el arreglo
34     |                         |   # move to the next position in the array
35     n += 1                   # aumentar el contador de datos ingresados
36     |                         |   # increase the entered data counter
37 ~ elif aux.isalpha():
38     cract.append(aux)        # si todos los caracteres son letras (ej: "hola", "abc")
39     |                         |   # if all characters are letters (e.g., "hola", "abc")
40     cract.append(aux)        # agregar el texto a la lista de caracteres
41     |                         |   # add the text to the characters list
42     n += 1                   # aumentar el contador de datos ingresados
43     |                         |   # increase the entered data counter
44
45 else: # si no es ni número ni solo letras (ej: "12a", "@", "1.5")
46     |                         |   # if it's neither number nor only letters (e.g., "12a", "@", "1.5")
47     print("No valido")       # mostrar mensaje de error
48     |                         |   # show error message
49
50 # mostrar resultados finales
51 # display final results
52 print(f"cantidad de numeros: {pos}") # total de números válidos guardados
53 |                         |   # total valid numbers stored
54 # print(f"Numeros ingresados: {num[:pos]}")
55 print(f"cantidad de caracteres: {len(cract)}") # total de cadenas de texto (solo letras) ingresadas
56 |                         |   # total text strings (only letters) entered
57 # print(f"String ingresados: {cract}")

```

```

38     cract.append(aux)        # if all characters are letters (e.g., "hola", "abc")
39     |                         |   # add the text to the characters list
40     n += 1                   # aumentar el contador de datos ingresados
41     |                         |   # increase the entered data counter
42
43 else: # si no es ni número ni solo letras (ej: "12a", "@", "1.5")
44     |                         |   # if it's neither number nor only letters (e.g., "12a", "@", "1.5")
45     print("No valido")       # mostrar mensaje de error
46     |                         |   # show error message
47
48 # mostrar resultados finales
49 # display final results
50 print(f"cantidad de numeros: {pos}") # total de números válidos guardados
51 |                         |   # total valid numbers stored
52 # print(f"Numeros ingresados: {num[:pos]}")
53 print(f"cantidad de caracteres: {len(cract)}") # total de cadenas de texto (solo letras) ingresadas
54 |                         |   # total text strings (only letters) entered
55 # print(f"String ingresados: {cract}")
56

```

Ilustración 9 Código Tarea 1

```

xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa5.py"
datos: asdef
datos: 23234
datos: ser
datos: 355
datos: gvgfdcx
datos: 34
datos: fdf
datos: 34
datos: sfr5
No valido
datos: 5t
No valido
datos: srt
datos: 453
cantidad de numeros: 5
cantidad de caracteres: 5
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>

```

Ilustración 10 Consola Tarea 1

Practica 5(resolución Tarea 1)

Descripción: En esta practica se le dio resolución al problema asignado en la tarea 1

```

1  # Inicializar arreglo de 10 elementos con valor -1
2  # Initialize array of 10 elements with value -1
3  arr = [-1]*10
4
5  # Inicializar lista vacía para caracteres
6  # Initialize empty list for characters
7  car = []
8  c = 0 # Contador para el número de entradas / Counter for number of inputs
9
10 # Bucle para capturar 10 entradas del usuario
11 # Loop to capture 10 user inputs
12 while (True):
13     a = input("Escribe un dato o valor: ")
14
15     # Verificar si la entrada es un número
16     # Check if input is a number
17     if a.isdigit():
18         arr[c] = int(a) # Almacenar número en el arreglo / Store number in array
19     # Verificar si la entrada es alfabética
20     # Check if input is alphabetic
21     elif a.isalpha():
22         car.append(a) # Agregar carácter a la lista / Add character to list
23         c += 1 # Incrementar contador de entradas / Increment input counter
24
25     # Salir del bucle después de 10 entradas
26     # Exit loop after 10 inputs
27     if c >= 10:
28         break
29
30 # Contar elementos válidos en el arreglo (no -1)
31 # Count valid elements in array (not -1)
32 c2 = 0
33 for i in arr:
34     if i != -1:
35         c2 += 1

```

```
35 | c2 += 1
36 |
37 # Mostrar resultados
38 # Display results
39 print(f"El arreglo tiene {c2}")
40 print(arr)
41 print(f"La lista tiene {len(car)}")
42 print(car)
```

Ilustración 11 Código Práctica 5

```
xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/sPrograma5.py"
Escribe un dato o valor: 45
Escribe un dato o valor: e
Escribe un dato o valor: 3
Escribe un dato o valor: fr
Escribe un dato o valor: 345
Escribe un dato o valor: f
Escribe un dato o valor: e
Escribe un dato o valor: 4
Escribe un dato o valor: 5
Escribe un dato o valor: f
El arreglo tiene 5
[45, -1, 3, -1, 345, -1, -1, 4, 5, -1]
La lista tiene 5
['e', 'fr', 'f', 'e', 'f']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> e
```

Ilustración 12 Consola Práctica 5

Practica 6(validación simple)

Descripción: Se utilizo validación básica (.isdigit / .isalpha) para determinar si un valor es digito o no, y según su clasificación se guarda en una lista o en un arreglo.

```

1  # Definicion de un metodo / Method definition
2  def resultado():
3      c2 = 0
4      print(f"La lista tiene {len(car)}")
5      # Contar elementos válidos en el arreglo (no -1)
6      # Count valid elements in array (not -1)
7      for i in arr:
8          if i != -1:
9              c2 += 1
10     print(f"El arreglo tiene {c2}")
11     print(arr)
12     print(car)
13
14 def hola():
15     c = 0
16     print("Hola mundo")
17     # Bucle principal para capturar 10 entradas
18     # Main loop to capture 10 inputs
19     while (True):
20         a = input("Escribe un dato o valor: ")
21
22         # Validar si es número y almacenar en arreglo
23         # Validate if number and store in array
24         if a.isdigit():
25             arr[c] = int(a)
26         # Validar si es texto y almacenar en lista
27         # Validate if text and store in list
28         elif a.isalpha():
29             car.append(a)
30             c += 1
31         # Condición de salida después de 10 entradas
32         # Exit condition after 10 inputs
33         if c >= 10:
34             break
35
36     resultado()
37
38     # variables globales o públicas / Global or public variables
39     # Arreglo pre-inicializado con valores -1
40     # Pre-initialized array with -1 values
41     arr = [-1]*10
42     # Lista vacía para caracteres / Empty list for characters
43     car = []
44
45
46     # llamado a metodo principal / Main method call
47     if __name__=="__main__":
48         hola()

```

Ilustración 13 Código Practica 6

```

Hola mundo
Escribe un dato o valor: 7
Escribe un dato o valor: jhjn
Escribe un dato o valor: 782
Escribe un dato o valor: 0123
Escribe un dato o valor: ooij
Escribe un dato o valor: kl8
Escribe un dato o valor: ana
Escribe un dato o valor: 1234
Escribe un dato o valor: 2345
Escribe un dato o valor: 345
La lista tiene 3
El arreglo tiene 6
[7, -1, 782, 123, -1, -1, 1234, 2345, 345]
['jhjn', 'ooij', 'ana']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> █

```

Ilustración 14 Consola Práctica 6

Practica 7(Guardar registros)

Descripción: Agregar nombre, edad, y sexo de 5 personas, y guardas todos los datos en una sola lista, manteniendo un orden coherente y lógico

```

1  ...
2  hacer un programa que lea nombre, edad y sexo de 5 personas, estos elementos tiene que estar dentro de una lista
3  make a program that reads name, age and gender of 5 people, these elements must be inside a list
4  ...
5  # .upper convierte a mayuscula / .upper converts to uppercase
6
7  # Lista para almacenar todos los registros de personas
8  # List to store all person records
9  registros = []
10 # persona = []
11 # def exper():
12 #     persona = {
13 #         "nombre": 'juan',
14 #         "edad": 12,
15 #         "sexo" : 'H'
16 #     }
17 #     print(persona)
18
19 def main():
20     # exper()
21     n = 0 # Contador de personas registradas / Person counter
22     # Bucle para registrar 5 personas / Loop to register 5 people
23     while (n < 5):
24         temp = input("Nombre: ")
25         # Validar que el nombre contenga solo letras
26         # Validate that name contains only letters
27         if temp.isalpha():
28             nombre = temp
29             temp = input("Edad: ")
30             # Validar que la edad sea un número
31             # Validate that age is a number
32             if temp.isdigit():
33                 edad = int(temp)
34                 temp = input("Sexo: ")
35                 # Validar que el sexo contenga solo letras
36                 # Validate that gender contains only letters
37                 if temp.isalpha():
38                     sex = temp

```

```

38     sex = temp
39
40         # Crear cadena con información de la persona
41         # Create string with person information
42         aux = "Nombre: " + nombre + ", Edad: " + str(edad) + ", Sexo: " + sex
43         # Agregar persona a la lista de registros
44         # Add person to the records list
45         registros.append(aux)
46         n += 1 # Incrementar contador de personas
47         # registros.pop # borra toda la lista / deletes the entire list
48     else:
49         print("Sexo no valido")
50     else:
51         print("Edad no valida")
52     else:
53         print("Nombre no valido")
54
55     # Mostrar todos los registros al final
56     # Display all records at the end
57     print(registros)
58
59
60
61 if __name__=="__main__":
62     main()

```

Ilustración 15 Código Práctica 7

```

xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa7.py"
Nombre: c
Edad: 112
Sexo: m
Nombre: a
Edad: 34
Sexo: f
Nombre: e
Edad: 23
Sexo: m
Nombre: f
Edad: 5
Sexo: m
Nombre: g
Edad: 43
Sexo: f
['Nombre: c, Edad: 112, Sexo: m', 'Nombre: a, Edad: 34, Sexo: f', 'Nombre: e, Edad: 23, Sexo: m', 'Nombre: f, Edad: 5, Sexo: m', 'Nombre: g, Edad: 43, Sexo: f']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>

```

Ilustración 16 Consola práctica 7

Practica 8 (Contador Letras)

Descripción: Detectar cuantos numero, cuantas mayúsculas, cuantas minúsculas, y cuantos espacios hay en una cadena que proporcionara el usuario

```

1  """
2  hacer un programa que lea una cadena y que muestre en pantalla
3  cuantos numero tiene, cuantas mayusculas, cuantas minusculas, y cuantos
4  espacios
5  make a program that reads a string and displays on screen
6  how many numbers it has, how many uppercase, how many lowercase, and how many
7  spaces
8  """
9  def main():
10     # Inicializar contadores / Initialize counters
11     mi = 0      # Contador de letras minúsculas / Lowercase letters counter
12     may = 0     # Contador de letras mayúsculas / Uppercase letters counter
13     c = 0       # Contador de números / Numbers counter
14     e = 0       # Contador de espacios / Spaces counter
15
16     # Cadena con todos los caracteres numéricos
17     # String with all numeric characters
18     num = "1234567890" # ["1","2","3","4","5","6","7","8","9","0"]
19
20     # Solicitar entrada al usuario / Prompt user for input
21     cadena = input("Escribe una cadena: \n")
22
23     # Analizar cada carácter de la cadena / Analyze each character in the string
24     for i in cadena:
25         # Verificar si es número / Check if it's a number
26         if i in num:
27             c += 1 # Incrementar contador de números / Increment numbers counter
28
29         # Verificar si es espacio / Check if it's a space
30         if i == ' ':
31             e += 1 # Incrementar contador de espacios / Increment spaces counter
32
33         # Verificar si es letra minúscula (códigos ASCII 97-122)
34         # Check if it's lowercase letter (ASCII codes 97-122)
35         if ord(i) >= 97 and ord(i) <= 122:
36             mi += 1 # Incrementar contador de minúsculas / Increment lowercase counter
37
38         # Verificar si es letra mayúscula (códigos ASCII 65-90)
39         # Check if it's uppercase letter (ASCII codes 65-90)
40         if ord(i) >= 65 and ord(i) <= 90:
41             may += 1 # Incrementar contador de mayúsculas / Increment uppercase counter
42
43     # Mostrar resultados / Display results
44     print(f"Los numeros son: {c}, y los espacion: {e}\n y las minusculas {mi}, y las mayusculas: {may}")
45
46
47
48
49
50 if __name__=='__main__':
51     main()

```

Ilustración 17 Código practica 8

```
xe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/practicaParcial1/programa8.py"
Escribe una cadena:
hola mundo
Los numeros son: 0, y los espacion: 1
y las minusculas 9, y las mayusculas: 0
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>
```

Ilustración 18 Consola practica 8

Tarea 2 (Identificar vocales, espacios, mayúsculas)

Descripción: Programa que verifica que no haya espacios, solo una mayúscula en el inicio, y que este todas las vocales

```

1 # hacer un programa que en una lista se introducan cadenas de caracteres, con las siguientes restricciones:
2 # make a program that inputs character strings into a list, with the following restrictions:
3 # -las cadenas no deben tener espacios / strings must not have spaces
4 # -la cadena solo puede tener mayúscula la primer letra / only the first letter can be uppercase
5 # -obligatoriamente debe tener todas las vocales. / must contain all vowels
6 # El programa no termina hasta que la lista tenga 5 elementos / program doesn't end until list has 5 elements
7 def main():
8     # crear una lista / create a list
9     lis = []
10    con = 0 # Contador de elementos válidos / Valid elements counter
11    while (con < 5):
12        es = True # Flag para espacios / Space flag
13        may = 0 # Contador de mayúsculas / Uppercase counter
14        est = False # Flag para estructura válida / Valid structure flag
15        voc = 0 # Contador de vocales encontradas / Vowel counter
16
17        # pedir cadena / request string
18        tem = input("Cadena: ")
19
20        # Analizar cada carácter de la cadena / Analyze each character in the string
21        for i in tem:
22            # Verificar si es mayúscula (códigos ASCII 65-90)
23            # Check if it's uppercase (ASCII codes 65-90)
24            if ord(i) >= 65 and ord(i) <= 90:
25                may += 1
26
27            # Verificar presencia de cada vocal (mayúscula o minúscula)
28            # Check presence of each vowel (uppercase or lowercase)
29            if ("A"in tem) or ("a"in tem):
30                voc += 1
31            if ("E"in tem) or ("e"in tem):
32                voc += 1
33            if ("I"in tem) or ("i"in tem):
34                voc += 1
35            if ("O"in tem) or ("o"in tem):
36                voc += 1
37            if ("U"in tem) or ("u"in tem):
38                voc += 1
39
40            # Verificar si hay espacios / Check for spaces
41            if " " in tem:
42                es = False
43
44            # Verificar estructura de mayúsculas:
45            # Check uppercase structure:
46            # - Solo una mayúscula y debe ser la primera letra
47            # - Only one uppercase and it must be the first letter
48            # - O ninguna mayúscula
49            # - Or no uppercase
50            if (may == 1) and (ord(tem[0]) >= 65 and ord(tem[0]) <= 90):
51                est = True
52            elif(may == 0):
53                est = True
54
55            # Verificar si cumple todas las condiciones / Check if all conditions are met
56            if est and (voc == 5) and es:
57                print("Valido")
58                con += 1
59                lis.append(tem)
60
61        # Mostrar lista final con todas las cadenas válidas
62        # Show final list with all valid strings

```

Ilustración 19 Código Tarea 2

```

62     # Show final list with all valid strings
63     print(lis)
64
65
66 ↘ if __name__ == "__main__":
67     main()

```

Ilustración 20 Código Tarea 2

```

Cadena: Aeiou
Valido
Cadena: Aeoijnja
Valido
Cadena: artiuU
Cadena: Aeioiuf
Valido
Cadena: Aeriuuiou
Valido
Cadena: Adesti
Cadena: Eoijjhdf
Cadena: Uosdjfjæiou
Valido
['Aeiou', 'Aeoijnja', 'Aeioiuf', 'Aeriuuiou', 'Uosdjfjæiou']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>

```

Ilustración 21 Consola Tarea 2

Practica 9 (Resolución Tarea 2)

Descripción: Se soluciono en clase el problema proporcionado para la tarea 2

```

1  # Función para verificar si una cadena contiene todas las vocales
2  # Function to check if a string contains all vowels
3 ↘ def vocales(cad):
4      # Bandera para cada vocal (inicialmente falsas)
5      # Flag for each vowel (initially false)
6      ba = False
7      be = False
8      bi = False
9      bo = False
10     bu = False
11
12     # Verificar presencia de cada vocal (mayúscula o minúscula)
13     # Check presence of each vowel (uppercase or lowercase)
14     if "a" in cad or "A" in cad:
15         ba = True
16     if "e" in cad or "E" in cad:
17         be = True
18     if "i" in cad or "I" in cad:
19         bi = True
20     if "o" in cad or "O" in cad:
21         bo = True
22     if "u" in cad or "U" in cad:
23         bu = True
24
25     # Si todas las vocales están presentes, agregar cadena a la lista
26     # If all vowels are present, add string to list
27     if ba and be and bi and bo and bu:
28         lista.append(cad)
29
30     # Mostrar estado actual de la lista
31     # Show current state of the list
32     print(f"La lista es: {lista}")
33
34     # Función para verificar que solo la primera letra es mayúscula
35     # Function to verify that only the first letter is uppercase
36 ↘ def minusculas(c1):
37     print(c1)
38     cm = 0 # Contador de letras minúsculas / Lowercase letters counter

```

```

38     cm = 0 # Contador de letras minúsculas / Lowercase letters counter
39
40     # Verificar que todos los caracteres excepto el primero son minúsculas
41     # Verify that all characters except the first are lowercase
42     for i in c1[1:]:
43         if ord(i) >= 97 and ord(i) <= 122: # Códigos ASCII para letras minúsculas
44             cm += 1 # ASCII codes for lowercase letters
45
46     # Si todos los caracteres después del primero son minúsculas
47     # If all characters after the first are lowercase
48     if cm == len(c1) - 1:
49         print(f"La cadena son minusculas excepto la primera{cm}")
50         vocales(cm) # Llamar a función de verificación de vocales / Call vowel check function
51     else:
52         print("Error la cadena no cumple")
53
54     # Función principal del programa
55     # Main program function
56     def main():
57         ce = 0 # Contador de caracteres no espacio / Non-space character counter
58         nc = "" # Cadena para almacenar caracteres no numéricos / String to store non-numeric characters
59
60         c = input("Escribe una cadena: \n")
61
62         # Contar caracteres que no son espacios
63         # Count characters that are not spaces
64         for i in c:
65             if ord(i) != 32: # 32 es código ASCII para espacio / 32 is ASCII code for space
66                 ce += 1
67
68         # Verificar si la cadena no contiene espacios
69         # Check if the string contains no spaces
70         if (ce == len(c)):
71             # Si la cadena es completamente alfabética
72             # If the string is completely alphabetic
73
73             # If the string is completely alphabetic
74             if c.isalpha():
75                 # Revisar que solo la primera letra sea mayúscula
76                 # Check that only the first letter is uppercase
77                 minusculas(c)
78             else:
79                 # Filtrar solo caracteres no numéricos
80                 # Filter only non-numeric characters
81                 for i in c:
82                     if ord(i) >= 48 and ord(i) <= 57: # Códigos ASCII para dígitos 0-9
83                         pass # ASCII codes for digits 0-9
84                     else:
85                         nc += i
86                 minusculas(nc) # Verificar la cadena filtrada / Check the filtered string
87             else:
88                 print("Error la cadena no cumple")
89
90         # Lista global para almacenar cadenas válidas
91         # Global list to store valid strings
92         lista = []
93
94         # Punto de entrada del programa
95         # Program entry point
96         if __name__ == "__main__":
97             # Bucle hasta que la lista tenga 5 elementos válidos
98             # Loop until the list has 5 valid elements
99             while(True):
100                 main()
101                 if len(lista) >= 5:
102                     break

```

Ilustración 22 Código Práctica 9

```
La cadena son minusculas excepto la primera4
La lista es: ['aeiua']
Escribe una cadena:
Aeiou1
Aeiou
La cadena son minusculas excepto la primera4
La lista es: ['aeiua', 'Aeiou']
Escribe una cadena:
Assdsdfnsemndkjsdx
Assdsdfnsemndkjsdx
La cadena son minusculas excepto la primera18
La lista es: ['aeiua', 'Aeiou']
Escribe una cadena:
Auiuou
Auiuou
La cadena son minusculas excepto la primera5
La lista es: ['aeiua', 'Aeiou']
Escribe una cadena:
aeuiuiu
aeuiuiu
La cadena son minusculas excepto la primera6
La lista es: ['aeiua', 'Aeiou']
Escribe una cadena:
aaaaeeeiiou
aaaaeeeiiou
La cadena son minusculas excepto la primera10
La lista es: ['aeiua', 'Aeiou', 'aaaaeeeiiou']
Escribe una cadena:
aaeiouuu
aaeiouuu
La cadena son minusculas excepto la primera7
La lista es: ['aeiua', 'Aeiou', 'aaaaeeeiiou', 'aaeiouuu']
Escribe una cadena:
aaaaeeeiiou
aaaaeeeiiou
La cadena son minusculas excepto la primera8
La lista es: ['aeiua', 'Aeiou', 'aaaaeeeiiou', 'aaeiouuu', 'aaaaeeeiiou']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> []
```

Ilustración 23 Consola Practica 9

Repaso 1(Cálculo de costos)

Descripción: Programa que calcula el costo de 5 productos proporcionados por el usuario, agregándole el valor de ganancia (Costo), mas su valor de IVA (16%)

```

29 def main():
30     # while(True):
31     # Bucle para procesar 5 productos / Loop to process 5 products
32     for i in range(0,5): # rango, tiene valor inicial y valor final / range has start and end value
33         # pedir nombre / request name
34         prod = input("Nombre producto: \n")
35         # pedir precio / request price
36         precioB = input("Precio primario: ")
37         # Validar que el precio sea un número válido
38         # Validate that the price is a valid number
39         if validar(precioB):
40             # calcular el precio final / calculate final price
41             # sumarle 12% mas 16% / add 12% plus 16%
42             costo = float(precioB) * 1.12 # Calcular costo con 12% / Calculate cost with 12%
43             preciof = costo * 1.16          # Calcular precio final con 16% IVA / Calculate final price with 16% VAT
44
45             # Mostrar resultados con formato de 2 decimales
46             # Show results with 2 decimal format
47             print(f"El producto: {prod}, tiene un costo de: {costo:.2f}, tiene un precio de venta de: {preciof:.2f}")
48             # ".2f" configura que el flotante solo tenga 2 decimales (los trunca)
49             # ":.2f" configures the float to have only 2 decimals (truncates them)
50         else:
51             print("Precio no valido ") # Mensaje de error / Error message
52
53         # resp = input("Desea otro numero (s/n) \n")
54
55         # if resp == "n" or resp == "N":
56         # if resp in ["n","N"]:
57         #     break
58
59     # Función para validar si una cadena puede convertirse a número
60     # Function to validate if a string can be converted to a number
61     def validar(num):
62         try:
63             nuevoN = float(num) # Intentar convertir a flotante / Try to convert to float
64             return True           # Retornar verdadero si es válido / Return true if valid
65
66         except:
67             return False          # Retornar falso si hay error / Return false if error
68
69     # Punto de entrada del programa
70     # Program entry point
71     if __name__ == "__main__":
72         main()
    
```

Ilustración 24 Código Repaso 1

```

SEMESTRE 3\estructura de datos\practica de python\practicaParcial1\repaso1.py"
Nombre producto:
Aqua
Precio primario: 56
El producto: Aqua,tiene un costo de: 62.72, tiene un precio de venta de: 72.76
Nombre producto:
123
Precio primario: 23
El producto: 123,tiene un costo de: 25.76, tiene un precio de venta de: 29.88
Nombre producto:
213
Precio primario: qw
Precio no valido
Nombre producto:
222
Precio primario: 3
El producto: 222,tiene un costo de: 3.36, tiene un precio de venta de: 3.90
Nombre producto:
2d
Precio primario: 12
El producto: 2d,tiene un costo de: 13.44, tiene un precio de venta de: 15.59
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> █
    
```

Ilustración 25 Consola Repaso 1

Repaso 2 (Formula general)

Descripción: se recrea la formula general, para así calcular una ecuación de segundo grado, verificando que los valores resultantes sean validos.

```

1 # Resolver ecuación cuadrática: ax^2 + bx + c = 0
2 # Solve quadratic equation: ax^2 + bx + c = 0
3 a = 1
4 b = 2
5 c = -15
6 p = 0
7 m = 0
8 r = 0
9 ra = 0.0
10 den = 0.0
11 x1 = 0.0
12 x2 = 0.0
13
14 # Calcular el discriminante (b^2 - 4ac)
15 # Calculate the discriminant (b^2 - 4ac)
16 p = b ** 2 # b al cuadrado / b squared
17 m = 4 * a * c # 4 veces a por c / 4 times a times c
18 r = p - m # Discriminante / Discriminant
19
20 # Verificar si el discriminante es positivo (raíces reales)
21 # Check if discriminant is positive (real roots)
22 if r > 0 :
23     print("Si se puede") # Se pueden calcular raíces reales / Real roots can be calculated
24     ra = r ** (1/2) # Calcular raíz cuadrada del discriminante / Calculate square root of discriminant
25     den = 2 * a # Calcular denominador común / Calculate common denominator
26     x1 = (-b + ra) / den # Primera raíz / First root
27     x2 = (-b - ra) / den # Segunda raíz / Second root
28     # Mostrar resultados con 2 decimales / Show results with 2 decimals
29     print(f"El valor de X1 = {x1:.2f}, y de X2: {x2:.2f}")
30 else:
31     print("No se puede") # No hay raíces reales / No real roots

```

Ilustración 26 Código Repaso 2

```

PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> & C:/Users
EMESTRE 3/estructura de datos/practica de python/practicaParcial1/repaso2.py"
Si se puede
El valor de X1 = 3.00, y de X2: -5.00
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1> []

```

Ilustración 27 Consola Repaso 2

Repaso 3 (Clasificaciones)

Descripción: Programa que almacena en una lista cualquier dato que le ingrese el usuario, pero conservando un tipo de dato original, mediante verificaciones, primero verificamos si es un float, luego si es un int, y si no es ninguno de los dos, entonces se trata como un String.

```

def valnum(item):
    n = 0.0
    if item.isdigit():
        print("Es int") # It's an integer
        lista.append(int(item)) # Add integer to the list / Agregar entero a la lista
        return True
    else:
        try:
            n = float(item)
            print("Es float") # It's a float
            lista.append(n) # Add float to the list / Agregar float a la lista
            return True
        except ValueError:
            return False # Not a number / No es un número

def leer():
    item = input("Dato: ") # Ask user for input / Pedir dato al usuario

    if valnum(item) == False:
        print("Es Str") # It's a string / Es una cadena
        lista.append(item) # Add string to the list / Agregar cadena a la lista

lista = [] # List to store all inputs / Lista para almacenar todos los datos

if __name__ == "__main__":
    while(True):
        leer() # Call the read function / Llamar función leer
        r = input("Desea agregar otro numero:(s/n)\n") # Ask if user wants to continue / Preguntar si quiere continuar

        if (r in ["n","N"]):
            print(lista) # Print the final list / Imprimir la lista final
            break

```

Ilustración 28 Código Repaso 3

```

Dato: weirjoierljdf
Es Str
Desea agregar otro numero:(s/n)
s
Dato: fhrieujhfoeildjs
Es Str
Desea agregar otro numero:(s/n)
n
['weirjoierljdf', 'fhrieujhfoeildjs']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>

```

Ilustración 29 Consola Práctica 3

Repaso 3 (Resolución clase)

```

1 def validar(a):
2     ne = 0
3     try:
4         ne = int(a) # Intentar convertir a entero / Try to convert to integer
5         return ne # Retornar entero / Return integer
6     except ValueError:
7         print("No es un entero") # No se pudo convertir a entero / Not an integer
8
9     try:
10        nf = float(a) # Intentar convertir a flotante / Try to convert to float
11        return nf # Retornar flotante / Return float
12    except ValueError:
13        print("No es un decimal") # No se pudo convertir a flotante / Not a float
14
15    return a # Si no es número, retornar como string / If not a number, return as string
16
17 def leer():
18     a = input("Escribe un dato \n") # Pedir dato al usuario / Ask the user for input
19     dato = validar(a) # Validar el dato ingresado / Validate the input
20     lista.append(dato) # Agregar el dato a la lista / Append the data to the list
21
22 lista = [] # Lista para almacenar los datos / List to store all inputs
23
24 if __name__ == "__main__":
25     while(True):
26         leer() # Llamar función leer / Call read function
27         res = input("Deseas otros/n: \n") # Preguntar si desea continuar / Ask if the user wants to continue
28         if res == "n" or res == "N":
29             print(lista) # Imprimir lista final / Print final list
30             break
31

```

Ilustración 30 Código Repaso 3

```

No es un entero
No es un decimal
Deseas otros/n:
s
Escribe un dato
we23
No es un entero
No es un decimal
Deseas otros/n:
w
Escribe un dato
2
Deseas otros/n:
Deseas otros/n:

Escribe un dato
2
Deseas otros/n:
2
Escribe un dato
n
No es un entero
No es un decimal
Deseas otros/n:
n
['tgjrfkedl', 2343, 'd', 'we23', 2, 2, 'n']
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\practicaParcial1>

```

Ilustración 31 Consola Repaso 3

Unidad 2

Practica 1

Descripción: En esta actividad se nos enseño como se definen y emplean las variables globales en Python, al mismo tiempo que observamos los principios de la recursividad, para ellos solicitamos 5 calificaciones al usuario, en lugar de emplear un bucle, volvemos a llamar a la función, guardando en progreso en una variable global

```

1  ^def inicio(num):
2      ^# global num # indico que voy a usar la variable publica num / I indicate that I will use the public variable num
3      ^ a = int(input("Escribe una calificación: \n")) # pide una calificación al usuario / asks the user for a grade
4      ^ num += 1 # incrementa el contador en 1 / increases the counter by 1
5      lista.append(a) # agrega la calificación a la lista / adds the grade to the list
6
7      if num >= 5: # si ya se han ingresado 5 calificaciones / if 5 grades have been entered
8          print(lista) # imprime la lista de calificaciones / prints the list of grades
9      else:
10         return inicio(num) # vuelve a llamar la función hasta llegar a 5 / calls the function again until reaching 5
11
12
13
14 lista = [] # lista vacía para guardar las calificaciones / empty list to store the grades
15 global num # declaro que es una variable global / declare that num is a global variable
16 num = 0 # defino num con valor inicial en 0 / define num with initial value 0
17
18 if __name__ == "__main__":
19     inicio(num) # llama a la función principal para iniciar / calls the main function to start
20

```

Ilustración 32 código Practica 1

```

PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2> & C:/Users/roc53/AppData/Local/Programs/Python/Python31
3/python.exe "c:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/Parcial 2/practica1_p2.py"
Escribe una calificación:
12
Escribe una calificación:
12
Escribe una calificación:
23
Escribe una calificación:
12
Escribe una calificación:
12
Escribe una calificación:
12
[12, 12, 23, 12, 12]
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2> 1

```

Ilustración 33 Salidas practica 1

Practica 2

Descripción: haciendo uso de la recursividad se solicitan 5 calificaciones al usuario, las cuales son agregadas a una lista, conservando su tipo de dato intacto, adicional a esto se obtiene el promedio de las 5 calificaciones y se muestra en la terminal. Para validar los datos se utilizó importo un archivo externo (elaborado por nosotros).

```

1  from p2_p2Validaciones import validaciones # Importa la clase validaciones desde otro archivo / Imports the class 'validaciones' f
2  val = validaciones() # Crea un objeto de la clase validaciones / Creates an object of the class 'validaciones'
3
4
5  class principal():
6      def __init__(self):
7          # constructor, primer código que siempre se ejecuta / constructor, first code that always runs
8          # aquí se cargan las variables necesarias / here the needed variables are initialized
9          self.lista = [] # lista vacía para guardar calificaciones / empty list to store grades
10         self.num = 0 # contador de cuántas calificaciones se han ingresado / counter of how many grades have been entered
11         self.a = "" # variable para guardar la calificación temporal / variable to temporarily store the grade
12
13
14     def inicio(self):
15         self.a = input("Escribe una calificación: \n") # pide una calificación al usuario / asks the user for a grade
16         if val.validarNumeros(self.a): # valida si el dato ingresado es un número / checks if the entered data is a number
17             self.num += 1 # incrementa el contador en 1 / increases the counter by 1
18             self.lista.append(int(self.a)) # convierte la entrada a entero y la agrega a la lista / converts the input to integer and adds it to the list
19
20         if self.num >= 5: # cuando ya se ingresaron 5 calificaciones / when 5 grades have been entered
21             print(self.lista) # imprime la lista de calificaciones / prints the list of grades
22             print(val.Promedio(self.lista)) # llama la función Promedio para calcular el promedio / calls Promedio function to calculate the average
23         else:
24             return self.inicio()
25             # se vuelve a llamar a sí misma hasta llegar a 5 / calls itself again until reaching 5
26             # ya no se necesita pasar parámetros porque las variables están en el objeto / no need to pass parameters since variables are in the object
27         else:
28             print('No es un numero') # si el dato no es número / if the input is not a number
29             self.inicio() # vuelve a pedir la calificación / asks again for the grade
30
31
32     if __name__ == "__main__":
33         app = principal() # crea un objeto de la clase principal / creates an object of the class 'principal'
34         app.inicio() # ejecuta el programa / runs the program
35

```

Ilustración 34 Código Practica 2

```

PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2> & 'c:\Users\roc53\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\roc53\vscode\extensions\ms-python.python-2025.14.0-win32-x64\bundled\libs\debugpy\launcher' '53381' '--' 'C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2\practica2_p2.py'
Escribe una calificación:
123
Escribe una calificación:
213
Escribe una calificación:
123
Escribe una calificación:
213
Escribe una calificación:
123
[123, 213, 123, 213, 123]
159.0
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2> 21

```

Ilustración 35 Salidas Practicas 2

Validaciones Practica 2

Descripción: Código que contiene un conjunto de validaciones, en esta ocasión se implementan la estructura de clases, para facilitar la organización y el fácil acceso a variables de cualquier función pertenecientes a esta misma clase. (Este fue el archivo que se importó para la práctica 2)

```

1  # English: Class for validations and calculations
2  # Español: Clase para validaciones y cálculos
3  class Validaciones():
4      # English: Constructor method to initialize instance variables
5      # Español: Método constructor para inicializar variables de instancia
6      def __init__(self):
7          # English: Variable to store the sum of numbers
8          # Español: Variable para almacenar la suma de números
9          self.suma = 0
10         # English: Variable to store the average
11         # Español: Variable para almacenar el promedio
12         self.prom = 0.0
13
14     # English: Method to validate if a string contains only digits
15     # Español: Método para validar si una cadena contiene solo dígitos
16     def validarNumeros(self, valor):
17         # English: Check if the string consists only of digits
18         # Español: Verifica si la cadena consiste solo de dígitos
19         if valor.isdigit():
20             # English: Return True if it's a valid number
21             # Español: Retorna True si es un número válido
22             return True
23         # English: Return False if it's not a valid number
24         # Español: Retorna False si no es un número válido
25         return False
26
27     # English: Method to calculate the average of a list of numbers
28     # Español: Método para calcular el promedio de una lista de números
29     def Promedio(self, lista):
30
31         def Promedio(self, lista):
32             # English: Iterate through each element in the list
33             # Español: Itera a través de cada elemento en la lista
34             for i in lista:
35                 # English: Accumulate the sum of all elements
36                 # Español: Acumula la suma de todos los elementos
37                 self.suma += i
38
39             # English: Calculate the average (sum divided by number of elements)
40             # Español: Calcula el promedio (suma dividida por el número de elementos)
41             self.prom = self.suma / len(lista)
42             # English: Return the calculated average
43             # Español: Retorna el promedio calculado
44             return self.prom

```

Ilustración 36 Código Validaciones Practica 2

Práctica 3:

Descripción: Comenzamos a utilizar ventana, mediante la implementación de la librería TKinter(); se nos enseñaron los principios básicos de este componente, como la asignación

del tamaño predeterminado, o la creación de componentes como botones, labels, y cajas de texto; en este caso específico se uso la propiedad .pack, para asignar un lugar a estos elementos en la ventana, su limitante principal es que solo permite un elemento por fila (o renglón).

```

1  from tkinter import *
2  from tkinter import messagebox # Importa el módulo para mostrar mensajes emergentes / Imports the module to show popup messages
3
4
5  def Ventana():
6
7      def revisar():
8          try:
9              u = str(us.get()) # obtiene el texto escrito en la caja de usuario / gets the text typed in the user entry
10             p = str(pas.get()) # obtiene el texto escrito en la caja de contraseña / gets the text typed in the password entry
11             if u == "" or p == "": # valida que no estén vacíos / checks if either is empty
12                 messagebox.showerror("Error","Faltan datos") # muestra error si faltan datos / shows error if data is missing
13             else:
14                 if u == "admin" and p == "12345": # validación simple de usuario y contraseña / simple user and password validation
15                     messagebox.showinfo("Validacion","Usuario y contrasela Correctos")
16                     # muestra mensaje de éxito / shows success message
17
18             else:
19                 messagebox.showerror('Error','Usuario y/o contraseña incorrectos')
20                 # muestra error si usuario o contraseña son incorrectos / shows error if user or password are wrong
21
22         except ValueError:
23             messagebox.showerror('Error','Introduce datos')
24             # en caso de que haya un error inesperado con los datos / in case of unexpected error with input
25
26
27     ven = Tk() # crea la ventana principal / creates the main window
28     ven.title('Programa 1 con Ventanas') # título de la ventana / window title
29     ven.geometry('400x200') # define el tamaño de la ventana / defines window size
30
31     # todo debe ir dentro de aquí / everything must go inside this window
32     # pack solo permite un elemento por renglón / pack only allows one element per row
33
34     Label(ven, text = "Usuario: ").pack(pady=10) # etiqueta para el usuario / label for username
35     us = Entry(ven) # caja de texto para usuario / input box for username
36     us.pack(pady=3)
37
38     Label(ven, text = "Password: ").pack(pady=10) # etiqueta para contraseña / label for password
39     pas = Entry(ven) # caja de texto para contraseña / input box for password
40     pas.pack(pady=5)
41
42     boton = Button(ven,text='Aceptar',command=revisar)
43     # botón que llama a la función revisar / button that calls revisar function
44     boton.pack(pady=3)
45
46     ven.mainloop() # mantiene la ventana abierta hasta que se cierre / keeps the window open until it is closed
47
48
49
50     if __name__=="__main__":
51         Ventana() # ejecuta la función principal / runs the main function

```

Ilustración 37 Código Práctica 3

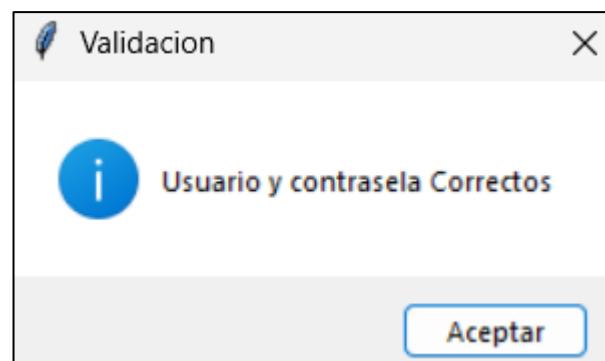
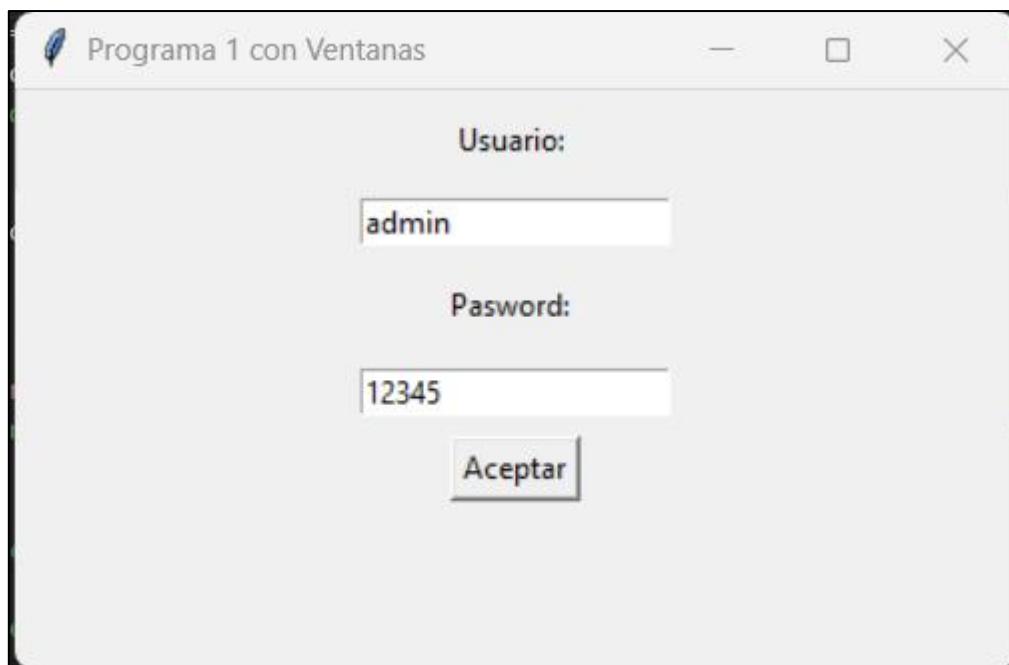


Ilustración 38 Salidas Práctica 3

Practica 4:

Descripción: Se puede considerar una versión mejorada de la practica 3, su funcionalidad es prácticamente la misma, con la gran diferencia que se incorporan validaciones, para asegurar que en las cajas de texto solo se acepten los formatos validos.

```

1  from tkinter import *
2  from tkinter import messagebox # Importa módulo para mostrar mensajes emergentes / imports module to show popup messages
3
4  class Ventana():
5      def __init__(self):
6          # Crear ventana principal / Create main window
7          self.ven = Tk()
8          self.ven.title('Programa 1 con Ventanas') # Título de la ventana / window title
9          self.ven.geometry('400x200') # Tamaño de la ventana / window size
10         # todo debe ir dentro de aquí / everything must go inside this window
11         # pack solo permite un elemento por renglón / pack only allows one element per row
12         # Label(ven, text = "Hola mundo").pack()
13         self.inicio() # Llama al método que crea los widgets / calls the method that creates the widgets
14
15     def inicio(self):
16         # Etiqueta y caja de texto para usuario / Label and entry for username
17         Label(self.ven, text = "Usuario: ").pack(pady=10)
18         self.us = Entry(self.ven)
19         self.us.pack(pady=3)
20
21         # Etiqueta y caja de texto para contraseña / Label and entry for password
22         Label(self.ven, text = "Password: ").pack(pady=10) # pady: espacio vertical / vertical padding
23         self.pas = Entry(self.ven)
24         self.pas.pack(pady=5)
25
26         # Botón que ejecuta la validación / Button that runs validation
27         self.boton = Button(self.ven, text='Aceptar', command=self.revisar)
28         self.boton.pack(pady=3)
29
30         self.ven.mainloop() # Mantiene la ventana activa hasta que se cierre / keeps window running until closed
31
32
33     def revisar(self):
34         # Función que valida los datos / Function that validates data
35         try:
36             self.u = str(self.us.get()) # Obtiene lo escrito en usuario / gets what was typed in username
37             self.p = str(self.pas.get()) # Obtiene lo escrito en password / gets what was typed in password
38
39             if self.u == "" or self.p == "": # Verifica que no estén vacíos / Checks if empty
40                 messagebox.showerror("Error", "Faltan datos")
41
42             else:
43                 # Validación básica de usuario y contraseña / Basic user & password validation
44                 if self.u == "admin" and self.p == "12345":
45                     messagebox.showinfo("Validacion", "Usuario y contrasela Correctos")
46                 else:
47                     messagebox.showerror('Error', 'Usuario y/o contraseña incorrectos')
48
49         except ValueError:
50             # Manejo de error si ocurre un problema inesperado / Handles unexpected input errors
51             messagebox.showerror('Error', 'Introduce datos')
52
53
54     if __name__ == "__main__":
55         app = Ventana() # Crea la instancia de la clase y lanza la app / Creates instance of the class and runs app
56

```

Ilustración 39 código Practica 4

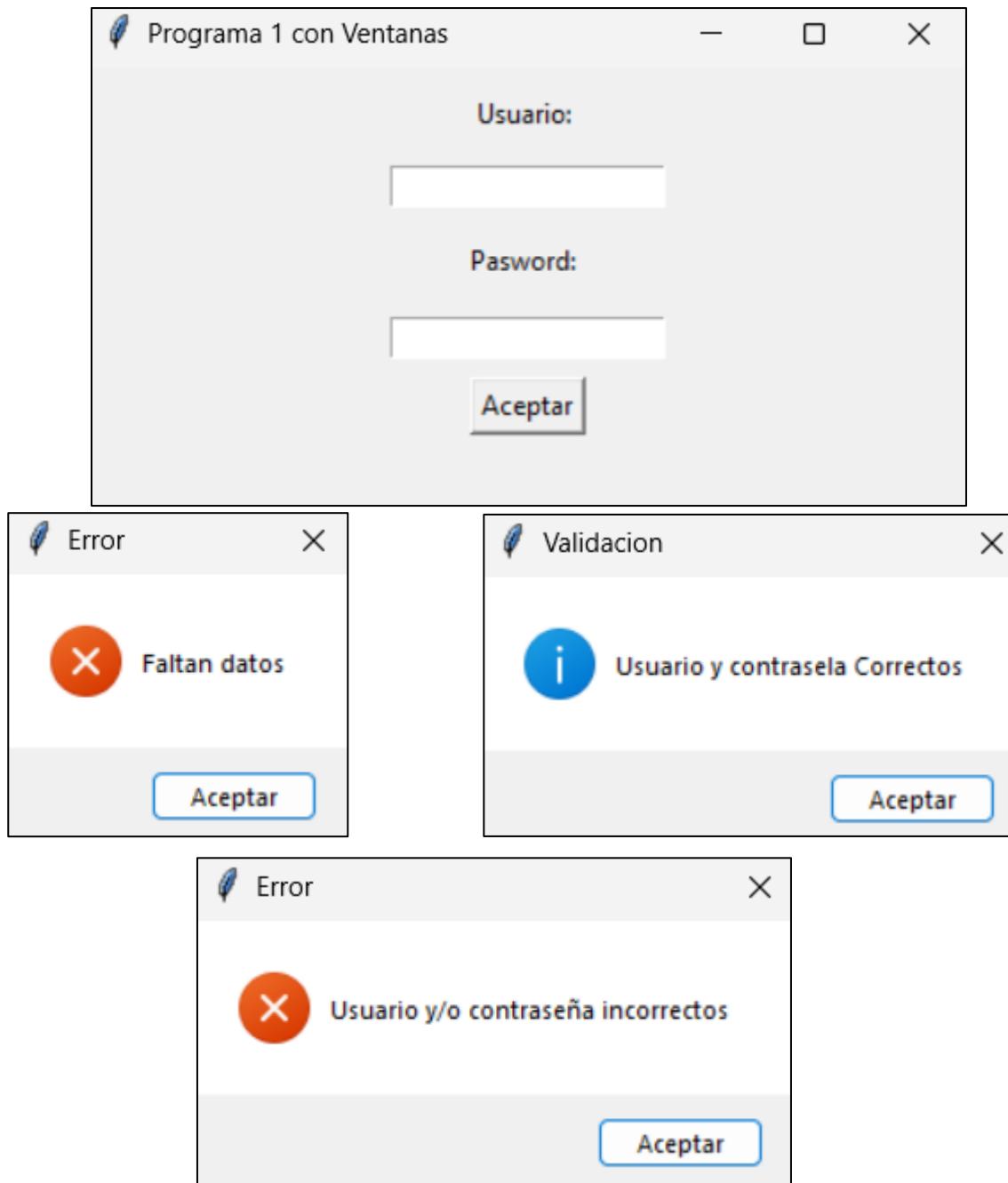


Ilustración 40 Salidas Práctica 4

Practica 5

Descripción: En esta actividad se solicitan 4 numero mediante una ventana, cuando el usuario presione el botón de promedio, se saca el promedio de estos número, los cuales se almacena en una lista, misma que se muestra mediante un label en la parte inferior de la ventana.

```

practica5_p2.py > ...
1  from tkinter import *
2  from tkinter import messagebox
3
4
5  class Principal():
6      def __init__(self): # █
7          # Crear ventana principal / Create main window
8          self.ven = Tk()
9          self.ven.title('Programa 5 con Ventanas') # Título de la ventana / Window title
10         self.ven.geometry('450x250') # Tamaño de la ventana / Window size
11         # todo debe ir dentro de aquí / Everything must go inside this window
12         # pack solo permite un elemento por renglón / pack only allows one element per row
13         # Label(ven, text = "Hola mundo").pack()
14         self.lista = [] # Lista donde se guardarán los promedios / List to store averages
15         self.inicio() # Llama al método que crea los widgets / Calls the method that creates the widgets
16
17  def inicio(self):
18      # Creación de etiquetas y campos de texto / Creating labels and text fields
19      l1 = Label(self.ven, text = "Escribe un numero: ")
20      l1.place(y=10,x=20) # y = filas, x= columnas / y = rows, x = columns
21      l2 = Label(self.ven, text="Escribe un numero")
22      l2.place(y=50,x=20)
23      self.n1 = Entry(self.ven)
24      self.n1.place(y=10, x= 130) # y = ☈ x = ☉
25      self.n2 = Entry(self.ven)
26      self.n2.place(y=50, x= 130)
27      l3 = Label(self.ven, text = "Escribe un numero: ")
28      l3.place(y=90,x=20) # Segunda fila de entradas / Second row of inputs
29      l4 = Label(self.ven, text="Escribe un numero")
30      l4.place(y=130,x=20)
31      self.n3 = Entry(self.ven)
32      self.n3.place(y=90, x= 130)
33      self.n4 = Entry(self.ven)
34      self.n4.place(y=130, x= 130)
35      l5 = Label(self.ven, text="Promedio")
36      l5.place(y=170,x=20)
37      self.l6 = Label(self.ven, text="xx") # Muestra el promedio actual / Displays current average
38      self.l6.place(y=170,x=130)
39      b1 = Button(self.ven, text="Promedio", command=self.promediar) # Botón para calcular promedio / Button to calculate average
40      b1.place(y= 50, x = 300)
41      b2 = Button(self.ven, text="Salir",command=self.salir) # Botón para cerrar programa / Button to exit program
42      b2.place(y= 90, x = 300)

43      self.l7 = Label(self.ven, text="[]") # Muestra la lista de promedios / Shows the list of averages
44      self.l7.place(y=200,x=200)
45      self.l8 = Label(self.ven, text="Promedio general 0.0") # Promedio acumulado / Overall average
46      self.l8.place(y=180,x=150)
47      self.ven.mainloop() # manda llamar ventana visual ☺ / launches the graphical window ☺

48
49  def promediar(self):
50      try:
51          # Obtener los valores de las cajas de texto / Get the values from text boxes
52          a = float(self.n1.get())
53          b = float(self.n2.get())
54          c = float(self.n3.get())
55          d = float(self.n4.get())
56          # Calcular promedio de los 4 números / Calculate average of the 4 numbers
57          pro = (a+b+c+d)/4
58          self.l6.config(text=str(pro)) # Mostrar promedio / Display average
59          self.lista.append(pro) # Guardar promedio en la lista / Save average in list
60          self.l7.config(text=str(self.lista)) # Mostrar lista actualizada / Display updated list
61          # Limpiar cajas de texto / Clear text boxes
62          self.n1.delete(0,END) # ☷ borra caja de texto / clears text box
63          self.n2.delete(0,END)
64          self.n3.delete(0,END)
65          self.n4.delete(0,END)
66          # Calcular promedio general / Calculate overall average
67          suma = self.sumar()
68          p = suma / (len(self.lista))
69          self.l8.config(text=f"Promedio: {str(p)}") # Mostrar promedio general / Show overall average

70
71      except ValueError:
72          # Si hay un error (por ejemplo texto no numérico) / If there's an error (like non-numeric input)
73          messagebox.showerror("Error","Algun dato no es numero") # Mensaje de error / Error message
74          # Limpiar cajas de texto / Clear text boxes
75          self.n1.delete(0,END)
76          self.n2.delete(0,END)
77          self.n3.delete(0,END)
78          self.n4.delete(0,END)
79
80
81

```

```

82     def salir(self):
83         self.ven.destroy() # Cierra la ventana / Closes the window
84
85         # self.ven.mainloop() # manda llamar ventana visual ☺ / launches graphical window ☺
86
87     def sumar(self):
88         # Suma todos los promedios guardados en la lista / Adds all averages stored in list
89         s = 0.0
90         for i in self.lista:
91             s += i
92
93         return s # Devuelve la suma total / Returns total sum
94
95
96
97
98 if __name__=="__main__":
99     app = Principal() # Crea y ejecuta la aplicación / Creates and runs the application
100

```

Ilustración 41 Código Práctica 5

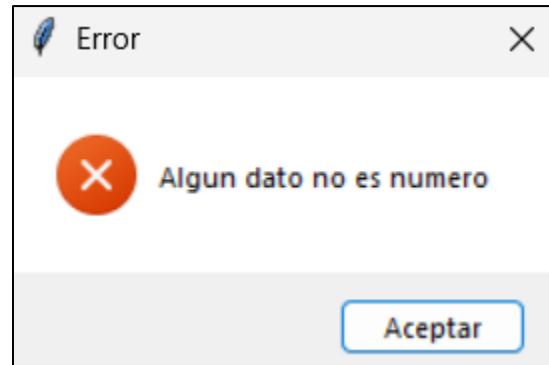
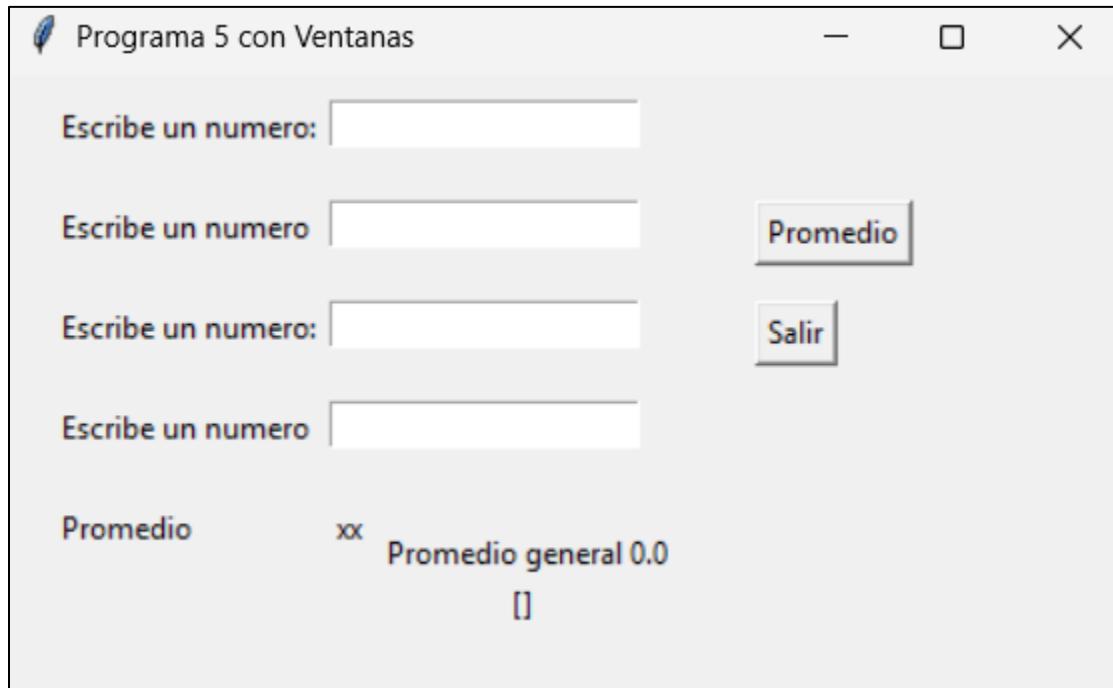


Ilustración 42 Salidas Práctica 5

Practica 6

Descripción: Programa que solicita en la ventana dos números, una vez agregados a la lista se muestran en la listBox posicionado en el costado derecho, después del primer registro los botones de mayor o menor , mostrar el valor mayor o menor respectivamente; finalmente el botón salir elimina la ventana.

```

1  from tkinter import *
2  from tkinter import messagebox
3
4  class Principal():
5      def __init__(self): # █
6          # Crear ventana principal / Create main window
7          self.ven = Tk()
8          self.ven.title('Programa 6 con Ventanas') # Título de la ventana / Window title
9          self.ven.geometry('450x350') # Tamaño de la ventana / Window size
10         # todo debe ir dentro de aquí / Everything must go inside this window
11         # pack solo permite un elemento por renglón / pack only allows one element per row
12         # Label(ven, text = "Hola mundo").pack()
13         self.lista = [] # Lista para almacenar los números ingresados / List to store entered numbers
14         # self.inicio() # Llama al método que crea los widgets / Calls the method that creates widgets
15         self.a = 0 # Variable auxiliar para primer número / Auxiliary variable for first number
16         self.b = 0 # Variable auxiliar para segundo número / Auxiliary variable for second number
17         self.lista = [] # Lista principal donde se guardan los valores / Main list storing values
18         self.aux1 = 0 # Auxiliar para buscar el mayor / Helper to find the largest
19         self.aux2 = 0 # Auxiliar para buscar el menor / Helper to find the smallest
20         self.con = 0 # Contador para el recorrido recursivo / Counter for recursive iteration
21
22     def inicio(self):
23         # Etiquetas y cajas de texto usando grid / Labels and text boxes using grid layout
24         # Label(self.ven,text="Programa 9").grid(row = 1, column=2)
25         l1 = Label(self.ven,text="Programa 9")
26         l1.grid(row = 1, column=2)
27         Label(self.ven,text=" ").grid(row = 2, column=2) # Espaciador visual / Visual spacer
28         l2 = Label(self.ven,text="Escribe un numero")
29         l2.grid(row = 3, column=1, padx=15, pady=10)
30         self.n1 = Entry(self.ven) # Caja de entrada para primer número / Input box for first number
31         self.n1.grid(row=3, column=2)
32         Label(self.ven,text=" ").grid(row = 4, column=2)
33         l3 = Label(self.ven,text="Escribe otro numero")
34         l3.grid(row = 5, column=1, padx=5, pady=5)
35         self.n2 = Entry(self.ven) # Caja de entrada para segundo número / Input box for second number
36         self.n2.grid(row=5, column=2)
37
38         # Botones con sus comandos / Buttons with their commands
39         b1 = Button(self.ven, text="Añadir", command=self.agregar)
40         b1.grid(row=6, column=1, pady=10)
41         b2 = Button(self.ven, text="Mayor", command=self.mayor)
42         b2.grid(row=6, column=2)

```

```

43     b3 = Button(self.ven, text="Menor", command=self.menor)
44     b3.grid(row=6, column=3)
45     b4 = Button(self.ven, text="Salir", command=self.salir)
46     b4.grid(row=6, column=4, padx=25)
47
48     # Etiquetas y lista visual / Labels and visual list
49     self.listaElem = Label(self.ven, text=" ") # Muestra lista completa / Shows the complete list
50     self.listaElem.grid(row = 8, column=2, pady=15)
51     self.listView = Listbox(self.ven, width= 15, bg= "black", activestyle= "dotbox", fg="white")
52     # Muestra los elementos agregados visualmente / Displays added elements visually
53     self.listView.grid(row=2, column=4)
54
55     self.ven.mainloop() # Inicia el ciclo principal de la interfaz / Starts GUI main loop
56
57 def salir(self):
58     self.ven.destroy() # destruimos ventana 🚫❌ / destroys the window 🚫❌
59
60 def mayor(self):
61     # Determina el número mayor de forma recursiva / Determines the largest number recursively
62     if len(self.lista) > 0:
63         if self.aux1 < self.lista[self.con]:
64             self.aux1 = self.lista[self.con]
65             self.con += 1
66         if len(self.lista) - 1 == self.con:
67             messagebox.showinfo("El Mayor", f"El mayor es: {self.aux1}") # Muestra el mayor / Shows largest
68             self.con = 0 # Reinicia el contador / Reset counter
69         else:
70             return self.mayor() # Llamada recursiva / Recursive call
71             #self.con += 1
72     else:
73         messagebox.showerror("Error", "Lista vacía") # Error si no hay elementos / Error if list empty
74
75 # def menor(self,n):
76 #     if self.aux > self.lista[n]
77 #     # (comentado) ejemplo de otra forma de calcular el menor / (commented) alternate way to calculate smallest
78
79 def menor(self):
80     # Determina el número menor de forma iterativa / Determines smallest number iteratively
81     if len(self.lista) > 0:
82
83         self.aux2 = self.lista[0] # Toma el primer valor como base / Takes first value as base
84         for i in self.lista:
85             if self.aux2 > i:
86                 self.aux2 = i
87             messagebox.showinfo("El Menor",f"El menor es {self.aux2}") # Muestra resultado / Displays result
88         else:
89             messagebox.showerror("Error", "Lista vacía") # Lista vacía / Empty list warning
90
91
92 def agregar(self):
93     # Agrega los números de las cajas de texto a la lista / Adds numbers from text boxes to list
94     try:
95         self.a = int(self.n1.get()) # Convierte primer número / Convert first number
96         self.lista.append(self.a)
97         self.listView.insert((self.listView.size() + 1),self.a) # Inserta en listbox / Insert into listbox
98         self.n1.delete(0,END) # Limpia entrada / Clear input box
99         self.b = int(self.n2.get()) # Convierte segundo número / Convert second number
100        self.lista.append(self.b)
101        self.listView.insert((self.listView.size() + 1),self.b) # Inserta en listbox / Insert into listbox
102        self.n2.delete(0,END) # Limpia entrada / Clear input box
103        self.listaElem.config(text=f"la lista: {self.lista}") # Actualiza etiqueta con lista / Update label with list
104        self.aux1 = self.lista[0] # Actualiza auxiliar inicial / update initial helper
105        self.listView.insert(self.a) # Inserta valor (parece redundante) / Inserts value (possibly redundant)
106
107        # print(self.lista) # Depuración opcional / Optional debug print
108
109    except ValueError:
110        messagebox.showerror("Error", "Algun dato no es numero") # Error si hay texto / Error if non-numeric input
111
112
113 if __name__ == "__main__":
114     app = Principal() # Crea el objeto principal / Create main app object
115     app.inicio() # Ejecuta la interfaz / Launch interface
116

```

Ilustración 43 Código Práctica 6

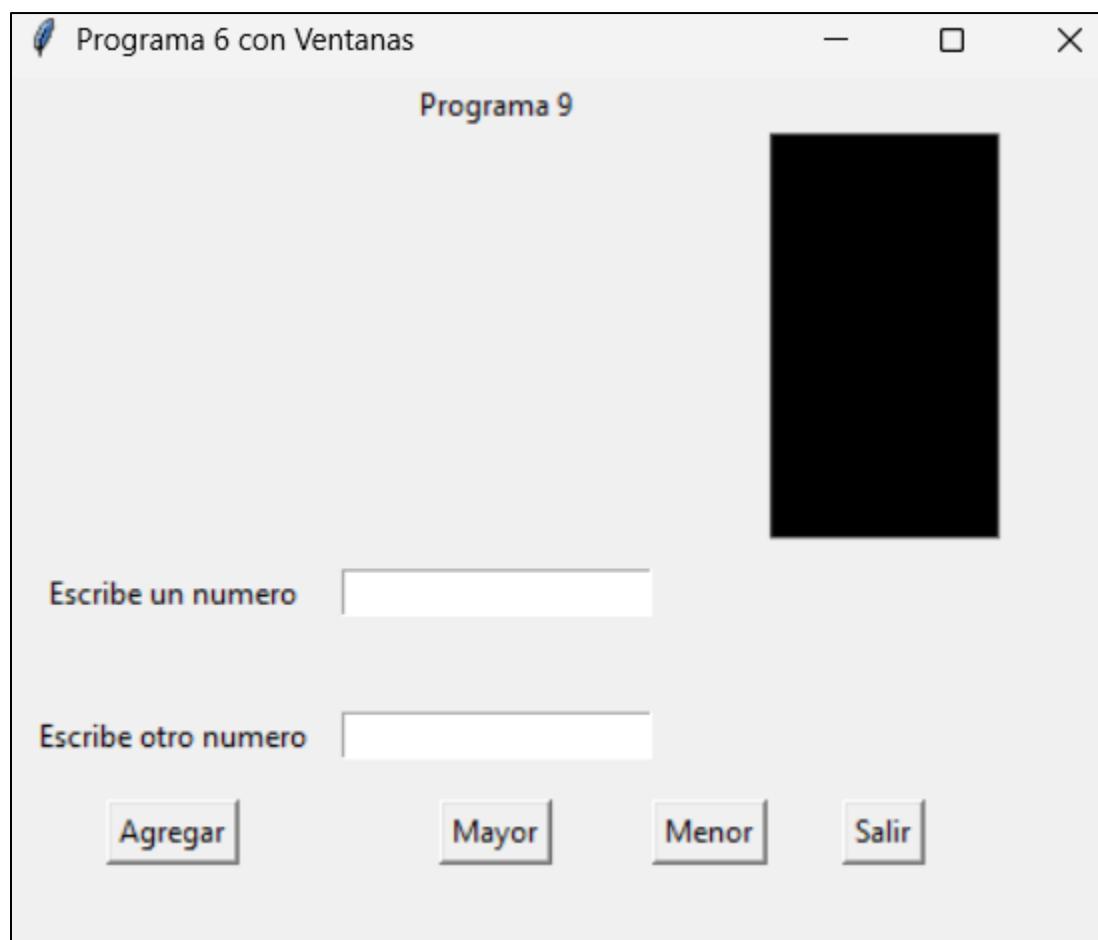


Ilustración 44 Salida Práctica 6

Practica 7

Descripción: Versión mejorada de la practica 6, sin embargo tiene cambios bastante notorios, primeramente el acomodo en lugar de ser mediante grid() para acomodar los elementos, se utilizo .place(), este asigna un ligar a los elementos en función a coordenadas, mientras que grid trataba la ventana como tabla. Eso si los datos que se ingresan no son tomados en cuenta ya que en su lugar se remplaza la entrada de datos por datos generados por la librería random, estos son números aleatorios.

```

1  from tkinter import *
2  from tkinter import messagebox
3  import random # importar librería random / import random library
4
5  class Principal():
6      def __init__(self): # █
7          # Crear ventana principal / Create main window
8          self.ven = Tk()
9          self.ven.title('Programa 10 con Ventanas') # Título de la ventana / Window title
10         self.ven.geometry('550x250') # Tamaño de la ventana / Window size
11         # todo debe ir dentro de aquí / Everything must go inside this window
12         # pack solo permite un elemento por renglón / pack only allows one element per row
13         # Label(ven, text = "Hola mundo").pack()
14         self.lista = [] # Lista donde se guardarán los números / List to store numbers
15         # self.inicio() # Llama al método que crea los widgets / Calls the method that creates widgets
16         self.a = 0 # Variable auxiliar / Helper variable
17         self.b = 0 # Variable auxiliar / Helper variable
18         self.lista = [] # Reinicia la lista / Reset list
19         self.aux1 = 0 # Auxiliar para buscar el mayor / Helper for max value
20         self.aux2 = 0 # Auxiliar para buscar el menor / Helper for min value
21         self.con = 0 # Contador para recorrido recursivo / Counter for recursion
22
23     def inicio(self):
24         # Crear etiquetas y campos de entrada / Create labels and entry fields
25         l1 = Label(self.ven, text="Programa 10")
26         l1.place(x = 200, y = 20)
27         l2 = Label(self.ven, text="Escribe un numero")
28         l2.place(x = 100, y = 50)
29         self.n1 = Entry(self.ven) # Caja de texto (no usada en este caso) / Text box (not used here)
30         self.n1.place(x = 100, y = 75)
31         l3 = Label(self.ven, text="Escribe otro numero")
32         l3.place(x = 250, y = 50)
33         self.n2 = Entry(self.ven) # Caja de texto (no usada en este caso) / Text box (not used here)
34         self.n2.place(x=250, y =75)
35
36         # Botones principales / Main buttons
37         b1 = Button(self.ven, text="Aregar", command=self.agregar) # Genera número aleatorio / Generate random number
38         b1.place(x=100, y =120)
39         b2 = Button(self.ven, text="Mayor", command=self.mayor) # Busca el mayor / Find largest number
40         b2.place(x=180, y =120)
41         b3 = Button(self.ven, text="Menor", command=self.menor) # Busca el menor / Find smallest number
42         b3.place(x=260, y =120)

```

```

43     b4 = Button(self.ven, text="Salir", command=self.salir) # Cierra ventana / Exit window
44     b4.place(x=340, y =120, width = 50)
45
46     # Etiquetas y lista visual / Labels and visual list
47     self.listaElem = Label(self.ven,text=" ") # Muestra el contenido de la lista / Shows list content
48     self.listaElem.place(x = 20, y =200)
49     self.listview = Listbox(self.ven, width= 15, bg= "black", activestyle= "dotbox", fg="white")
50     # Listbox donde se muestran los valores agregados / Listbox showing added values
51     self.listview.place(x=420, y = 20)
52     self.ven.mainloop() # Inicia el ciclo de eventos de la interfaz / Starts GUI main loop
53
54     def salir(self):
55         self.ven.destroy() # destruimos ventana * ⚡ / destroys the window * ⚡
56
57     def mayor(self):
58         # Calcula el número mayor (versión recursiva) / Calculates largest number (recursive version)
59         if len(self.lista) > 0:
60             if self.aux1 < self.lista[self.con]:
61                 self.aux1 = self.lista[self.con]
62             self.con += 1
63             if len(self.lista) -1 == self.con:
64                 messagebox.showinfo("El Mayor", f"El mayor es: {self.aux1}") # Muestra resultado / Show result
65                 self.con = 0 # Reinicia contador / Reset counter
66             else:
67                 return self.mayor() # Llamada recursiva / Recursive call
68         else:
69             messagebox.showerror("Error", "Lista vacía") # Si no hay datos / If no data present
70
71     # def menor(self,n):
72     #     if self.aux > self.lista[n]
73     # (comentado) Ejemplo alternativo de función / (commented) Alternate function example
74
75     def menor(self):
76         # Calcula el número menor (versión iterativa) / Calculates smallest number (iterative version)
77         if len(self.lista) > 0:
78             self.aux2 = self.lista[0] # Toma el primer valor como base / Take first value as base
79             for i in self.lista:
80                 if self.aux2 > i:
81                     self.aux2 = i
82             messagebox.showinfo("El Menor",f"El menor es {self.aux2}") # Muestra resultado / Show result
83
84     else:
85         messagebox.showerror("Error", "Lista vacía") # Error si no hay elementos / Error if list is empty
86
87
88     def agregar(self):
89         try:
90             # self.a = int(self.n1.get()) # Código original (comentado) / Original code (commented)
91             self.a = random.randint(1,100) # Genera número aleatorio / Generate random number
92             self.lista.append(random.randint(1,100)) # Agrega número aleatorio distinto / Add another random number
93             self.listview.insert((self.listview.size() + 1),self.a) # Inserta número en la lista visual / Insert into visual list
94             # self.n1.delete(0,END)
95             # self.b = int(self.n2.get())
96             # self.lista.append(self.b)
97             # self.listview.insert((self.listview.size() + 1),self.b)
98             # self.n2.delete(0,END)
99
100            self.listaElem.config(text=f"La lista: {self.lista}") # Actualiza texto con lista / Update label with list
101            self.aux1 = self.lista[0] # Reinicia auxiliar / Reset helper
102            self.listview.insert(self.a) # Inserta elemento (parece redundante) / Insert element (possibly redundant)
103
104            # print(self.lista) # Para depurar valores / For debugging values
105
106        except ValueError:
107            messagebox.showerror("Error", "Algun dato no es numero") # Error si el dato no es válido / Error if input invalid
108
109
110    if __name__ == "__main__":
111        app = Principal() # Crea objeto principal / Create main app object
112        app.inicio() # Inicia interfaz / Start interface
113

```

Ilustración 45 Código Práctica 7

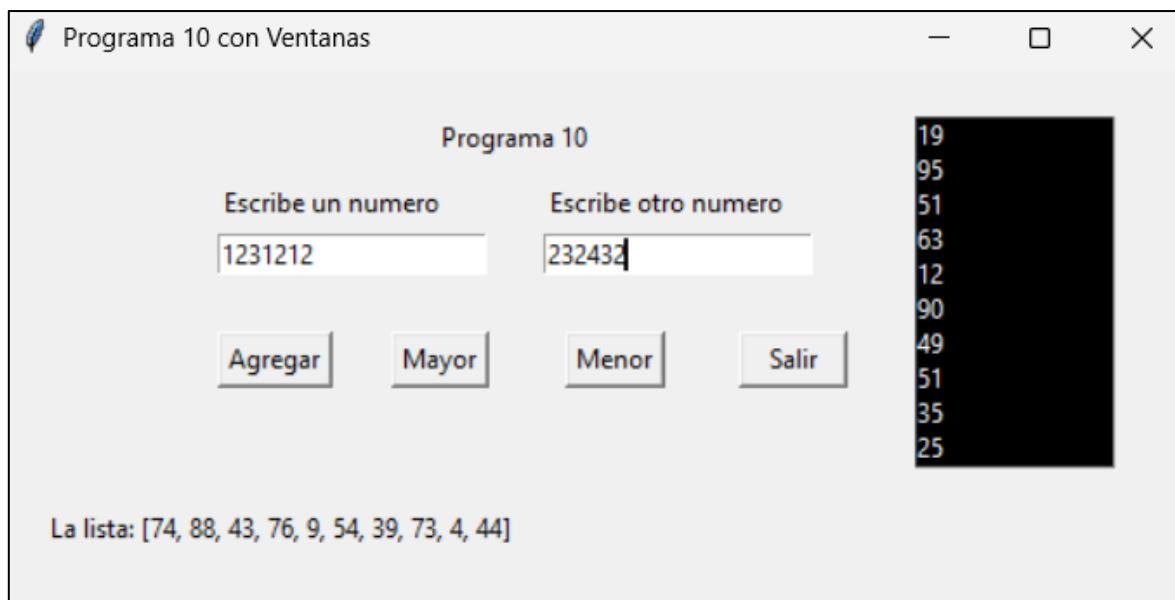


Ilustración 46 Salidas Práctica 7

Repaso 1

Descripción: La práctica tiene como objetivo practicar las diversas formas de validar un mismo tipo de dato, algunas de estas formas están contenidas en el archivo de validaciones de repaso, esto se logra ingresando datos mediante una caja de texto y presionando el botón validar, este a su vez llama a la función validar_dat, la cual a su vez llama al archivo de validaciones mediante un objeto de este archivo (librería)

```

1  v from tkinter import *
2  from tkinter import messagebox
3  from ValidarReposo import Validaciones # █ Importa la clase Validaciones desde otro archivo / imports the Validaciones class from another file
4
5  v class Principal():
6      def __init__(self): # solo se ejecuta una vez / runs only once
7          self.ventana = Tk() # █ Crea la ventana principal / creates the main window
8          self.ventana.geometry("500x300") # tamaño de ventana / sets window size
9          self.lista = [] # █ Lista para guardar los datos ingresados / list to store entered data
10         self.valid = Validaciones() # █ Crea un objeto de la clase Validaciones / creates an instance of Validaciones class
11
12 v     def inicio(self):
13         # █ Etiquetas (Labels) de texto en la ventana / Text labels in the window
14         Label(self.ventana, text="Programa de python con TKinter").place(x = 100, y = 20)
15         Label(self.ventana, text="Escribe un dato: ").place(x = 50, y = 60)
16
17         # █ Campo de texto para escribir datos / Text input field
18         self.dato = Entry(self.ventana)
19         self.dato.place(x = 150, y = 60, width= 150) # height(largo) | width (ancho)
20
21         # █ Botón que valida el dato ingresado / Button that validates entered data
22         Button(self.ventana, text="Validar", command = self.validar_dat).place(x = 150, y = 90, width= 150)
23
24         # █ Etiqueta para mostrar resultados / Label to display results
25         self.mostrar = Label(self.ventana, text="")
26         self.mostrar.place(x = 20, y = 130)
27
28         # █ Inicia el bucle principal (muestra la ventana) / Starts main event loop (shows the window)
29         self.ventana.mainloop()
30
31 v     def validar_dat(self):
32         val = self.dato.get() # █ Obtiene el dato del Entry / Gets the value from input field
33
34         if val != "": # █ Verifica que no esté vacío / Checks that it's not empty
35             self.revisar(val) # █ Llama a revisar (solo imprime) / Calls revisar (only prints)
36             self.lista.append(val) # █ Agrega el dato a la lista / Adds data to the list
37             self.dato.delete(0, END) # █ Borra el texto del Entry / Clears the input field
38
39             # Actualiza la etiqueta con la lista actual / Updates label with current list
40             self.mostrar.config(text= f"La lista es: {self.lista}")
41
42 v         # █ Llamadas posibles a distintos métodos de validación / Possible calls to different validation methods
43         # respuesta = self.valid.valASCII(val)
44         # respuesta = self.valid.valCError(val)
45         respuesta = self.valid.validarConStr(val) # █ Usa método validarConStr / Uses validarConStr method
46
47         # █ Muestra el resultado de la validación / Shows validation result
48         messagebox.showinfo("Validar datos", f"El dato es: {respuesta}")
49
50     else:
51         # █ Muestra error si no se escribió nada / Shows error if entry is empty
52         messagebox.showerror("Error", "La caja de texto esta vacia")
53
54     def revisar(self,v):
55         print(v) # █ Imprime el valor recibido (solo para depuración) / Prints received value (for debugging)
56
57     # █ Punto de entrada del programa / Entry point of the program
58     if __name__ == "__main__":
59         app = Principal() # █ Crea el objeto principal / Creates main object
60         app.inicio() # █ Llama al método que inicia la interfaz / Calls the method that starts the GUI
61         pass # █ No hace nada, solo marca fin del bloque / Does nothing, placeholder
62

```

Ilustración 47 Código Repaso 1

```

1 class Validaciones():
2     def __init__(self):
3         # Inicializa un índice que se usará en validaciones recursivas / Initialize an index for recursive validation
4         self.index = 0
5
6     def valASCII(self, valor):
7         # Contadores para números y letras / Counters for numbers and letters
8         con = 0
9         con2 = 0
10
11        # Recorre cada carácter del valor / Loop through each character in the value
12        for i in valor:
13            # Si el código ASCII está entre 48 y 57, es un número / If ASCII code is between 48-57, it's a number
14            if ord(i) >= 48 and ord(i) <= 57:
15                con += 1
16            # Si el código ASCII está entre 65-90 o 97-122, es una letra / If ASCII code is between 65-90 or 97-122, it's a letter
17            if (ord(i) >= 65 and ord(i) <= 90) or (ord(i) >= 97 and ord(i) <= 122):
18                con2 += 1
19
20        # Si todos son números / If all characters are numbers
21        if con == len(valor):
22            return "Numeros"
23        # Si todos son letras / If all characters are letters
24        elif con2 == len(valor):
25            return "Letras"
26
27        # Si contiene mezcla de letras y números / If it contains both letters and numbers
28        return "Letras y numeros"
29
30    def valCError(self, valor):
31        # Variables temporales para convertir el valor / Temporary variables to convert value
32        a = 0
33        b = 0.0
34        try:
35            # Intenta convertir a entero / Try to convert to integer
36            a = int(valor)
37            return "Numeros"
38
39        except ValueError:
40            try:
41                # Si no es entero, intenta convertir a decimal / If not integer, try to convert to float
42                b = float(valor)
43                return "Decimales"
44
45            except ValueError:
46                # Si no se puede convertir, entonces contiene letras o símbolos / If neither, it has letters or symbols
47                return "Letras o numeros"
48
49
50        # Este return es redundante, pero no afecta la ejecución / This return is redundant but harmless
51        return "Letras o numeros"
52
53    def validarConStr(self, valor):
54        # Imprime el valor para depuración / Print the value for debugging
55        print(valor)
56
57        # Revisión recursiva carácter por carácter / Recursive check character by character
58        if self.index < len(valor):
59            # Si encuentra '@', se asume que es un correo / If '@' is found, assume it's an email
60            if valor[self.index] == '@':
61                self.index = 0 # Reinicia el índice para futuras llamadas / Reset index for future calls
62                return "si es un correo"
63            else:
64                # Si no, avanza al siguiente carácter / Otherwise, move to next character
65                if self.index < len(valor):
66                    self.index += 1
67                    return self.validarConStr(valor) # Llamada recursiva / Recursive call
68                else:
69                    self.index = 0
70                    return "no es un correo"
71
72            # Si se llega al final sin encontrar '@' / If the end is reached without finding '@'
73            self.index = 0
74            return "no es un correo"

```

Ilustración 48 Código de las Validaciones del Repaso1

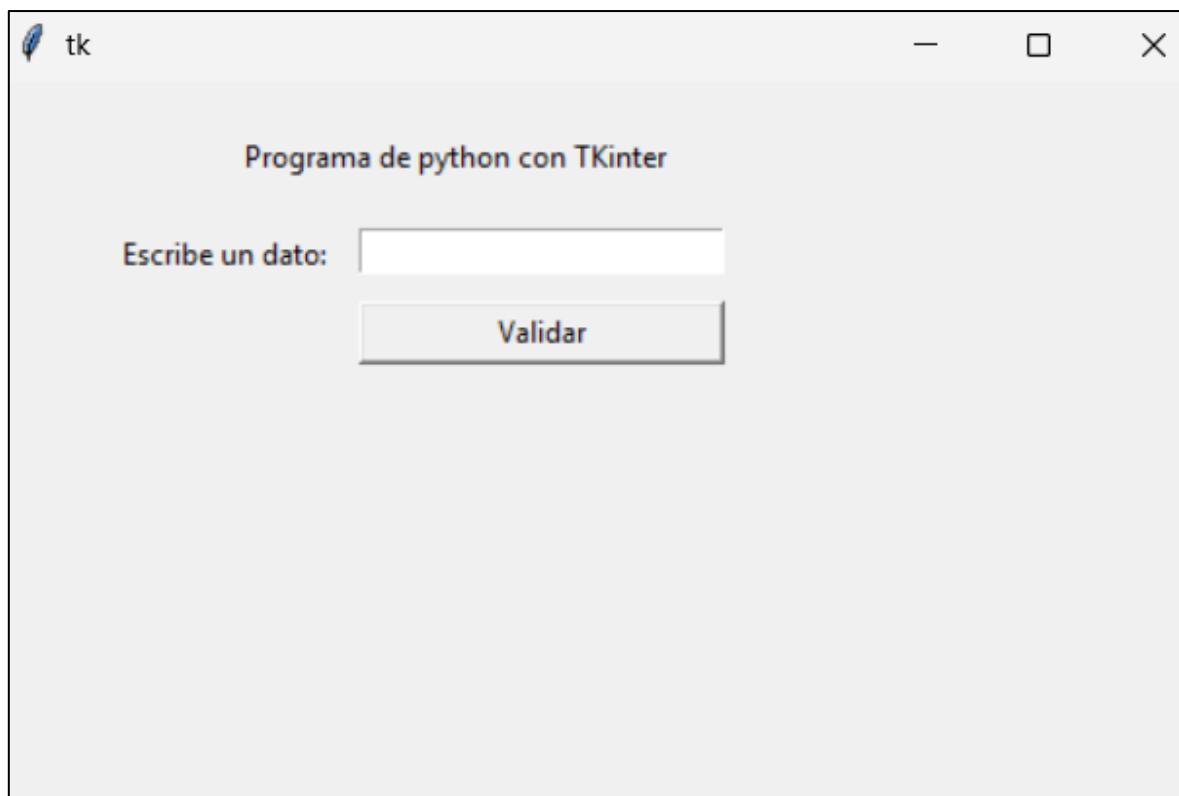


Ilustración 49 Salida Repaso 1

Actividad Clase (Resolución Examen Parcial 1):

Descripción: Con la finalidad de aclarar dudas, se soluciono el examen del parcial 1, el cual consistía en: 'hacer programa que lea nombre y 3 calificaciones que se agregara a una lista con las siguientes restricciones: las calificaciones se agregarán en orden descendente, una vez registrado esto se realizará una pregunta, si desea agregar otra persona, si la respuesta es si, la primer lista se agregará a otra lista que contenga los 4 datos, para agregar otra persona.'

```

1  '''hacer programa que lea nombre y 3 calificaciones que se agregara a una lista con las siguientes restricciones:
2  ~ las calificaciones se agregaran en orden descendente, una vez registrado esto se realizara una pregunta
3  si desea agregar otra persona, si la respuesta es si, la primer lista se agregara a otra lista que contenga los
4  datos, para agregar otra persona'''
5
6
7 def inicio():
8     # Initialize an empty list to store the current person's data (name and sorted grades)
9     # Inicializa una lista vacía para almacenar los datos de la persona actual (nombre y calificaciones ordenadas)
10    lista1 = []
11
12    # Prompt user to input a name
13    # Solicitud al usuario que ingrese un nombre
14    nom = input('Escribe un nombre')
15
16    # Prompt user to input three grades as integers
17    # Solicitud al usuario que ingrese tres calificaciones como enteros
18    c1 = int(input("Escribe una calificación"))
19    c2 = int(input("Escribe una calificación"))
20    c3 = int(input("Escribe una calificación"))
21
22    # Compare the three grades to sort them in descending order manually
23    # Compara las tres calificaciones para ordenarlas manualmente en orden descendente
24    if (c1 > c2):
25        if (c1 > c3):
26            # c1 is the highest grade
27            # c1 es la calificación más alta
28            print(f"es mayor {c1}")
29            lista1.append(nom)
30            lista1.append(c1)
31            if (c2 > c3):
32                # c2 is middle, c3 is lowest
33                # c2 es la del medio, c3 es la más baja
34                print(f"es el de enmedio {c2}")
35                print(f"es el menor {c3}")
36                lista1.append(c2)
37                lista1.append(c3)
38                lista2.append(lista1)
39            else:
40                # c3 is middle, c2 is lowest
41                # c3 es la del medio, c2 es la más baja
42                print(f"es el de enmedio{c3}")

```

```

43     print(f"el menor es {c2}")
44     lista1.append(c3)
45     lista1.append(c2)
46     lista2.append(lista1)
47
48     else:
49         # c3 is highest, c1 is middle, c2 is lowest
50         # c3 es la más alta, c1 es la del medio, c2 es la más baja
51         print(f"es el de enmedio {c1}")
52         print(f"es el mayor {c3}")
53         print(f"es el menor {c2}")
54         lista1.append(nom)
55         lista1.append(c3)
56         lista1.append(c1)
57         lista1.append(c2)
58         lista2.append(lista1)
59
60     else:
61         if (c2 > c3):
62             # c2 is the highest grade
63             # c2 es la calificación más alta
64             print(f"Es mayor {c2}")
65             lista1.append(nom)
66             lista1.append(c2)
67             if (c1 > c3):
68                 # c1 is middle, c3 is lowest
69                 # c1 es la del medio, c3 es la más baja
70                 print(f"El de en medio es {c1}")
71                 print(f"es el menor {c3}")
72                 lista1.append(c1)
73                 lista1.append(c3)
74                 lista2.append(lista1)
75             else:
76                 # c3 is middle, c1 is lowest
77                 # c3 es la del medio, c1 es la más baja
78                 print(f"es el de enmedio {c3}")
79                 print(f"es el menor {c1}")
80                 lista1.append(c3)
81                 lista1.append(c1)
82                 lista2.append(lista1)
83
84     else:
85         # c3 is highest, c2 is middle, c1 is lowest
86         # c3 es la más alta, c2 es la del medio, c1 es la más baja
87         print(f' es el de en medio {c2}')
88         print(f' es mayor {c3}')
89         print(f' es el de en menor {c1}')
90         lista1.append(nom)
91         lista1.append(c3)
92         lista1.append(c2)
93         lista1.append(c1)
94         lista2.append(lista1)
95
96     # Global list to store all registered persons with their sorted grades
97     # Lista global para almacenar todas las personas registradas con sus calificaciones ordenadas
98     lista2 = []
99
100    if __name__ == "__main__":
101        # Infinite loop to keep registering people until user decides to stop
102        # Bucle infinito para seguir registrando personas hasta que el usuario decida detenerse
103        while(True):
104            # Call the function to register a new person
105            # Llama a la función para registrar una nueva persona
106            inicio()
107
108            # Ask user if they want to add another person
109            # Pregunta al usuario si desea agregar otra persona
110            a = input("desea agregar otra persona s/n")
111
112            # If user inputs any form of "no", print the final list and exit
113            # Si el usuario ingresa cualquier forma de "no", imprime la lista final y sale
114            if a in ("n,N,NO,no,No"):
115                print(lista2)
                break

```

Ilustración 50 Código solución de examen

```
:/Users/roc53/Documents/SEMESTRE 3/estructura de datos/practica de python/Parcial 2/examenP1.py"
Escribe un nombreabrdres
Escribe una calificacion123
Escribe una calificacion1234
Escribe una calificacion12
Escribe una calificacion1234
El de en medio es 123
es el menor 12
desea agregar otra persona s/n
[['abrdres', 1234, 123, 12]]
PS C:\Users\roc53\Documents\SEMESTRE 3\estructura de datos\practica de python\Parcial 2> █
```

Ilustración 51 Salidas de solución de examen

Tarea 1 (Solución de examen Parcial 1)

Descripción: se traslada la funcionalidad de la resolución del examen parcial 1. Pero con el desafío de replicarlo en una ventana.

```
1  '''hacer programa que lea nombre y 3 calificaciones que se agregara a una lista con las siguientes restricciones:
2  las calificaciones se agregarán en orden descendente, una vez registrado esto se realizará una pregunta
3  si desea agregar otra persona, si la respuesta es si, la primera lista se agregará a otra lista que contenga los
4  datos, para agregar otra persona'''
5
6 from tkinter import *
7 from tkinter import messagebox
8
9
10 class Ventana():
11     def __init__(self):
12         self.lista = [] # █ lista principal donde se guardarán todos los registros / main list to store all records
13         self.ven = Tk() # █ crea la ventana principal / creates the main window
14         self.ven.title("Registro de calificación") # █ título de la ventana / window title
15         self.ven.geometry("400x200") # █ tamaño de la ventana / window size
16         self.inicio() # █ llama a la función que dibuja los widgets / calls function to draw widgets
17
18     def inicio(self):
19         # █ Etiqueta y entrada para el nombre / Label and entry for name
20         Label(self.ven, text = "Nombre: ").pack(pady = 10)
21         self.name = Entry(self.ven)
22         self.name.pack(pady = 3)
23
24         # █ Etiquetas y entradas para calificaciones / Labels and entries for grades
25         Label(self.ven, text = "Calificación 1: ").pack(pady = 10)
26         self.cal1 = Entry(self.ven)
27         self.cal1.pack(pady = 3)
28
29         Label(self.ven, text = "Calificación 2: ").pack(pady = 10)
```

```

30     self.cal2 = Entry(self.ven)
31     self.cal2.pack(pady = 3)
32
33     Label(self.ven, text = "Calificación 3: ").pack(pady = 10)
34     self.cal3 = Entry(self.ven)
35     self.cal3.pack(pady = 3)
36
37     # ● Botón para agregar otro registro / Button to add another record
38     self.botonAg = Button(self.ven, text = "Agregar otro", command = self.estruct)
39     self.botonAg.pack(pady = 2)
40
41     # ■ Botón para salir y mostrar lista final / Button to exit and show final list
42     self.botonsalir = Button(self.ven, text = "Salir", command = self.fin)
43     self.botonsalir.pack(pady = 2)
44
45     self.ven.mainloop() # ☰ mantiene abierta la ventana / keeps window open
46
47 def fin(self):
48     self.estruct() # 📁 guarda el último registro antes de salir / saves last record before exiting
49     messagebox.showinfo("Registro",self.lista) # 📜 muestra lista final / shows final list
50     exit(0) # 🔁 cierra el programa / closes program
51
52 def estruct(self):
53     temlis = [] # ✎ lista temporal para un registro (nombre + calificaciones) / temp list for one record
54
55     try :
56         # 📂 obtiene valores de las entradas / gets values from entries
57
58         self.nam = str(self.name.get())
59         self.c1 = int(self.cal1.get())
60         self.c2 = int(self.cal2.get())
61         self.c3 = int(self.cal3.get())
62
63         # 📑 agrega los datos a la lista temporal / adds data to temporary list
64         temlis.append(self.nam) # agrega nombre / adds name
65         temlis.append(self.c1) # agrega calificación 1 / adds grade 1
66         temlis.append(self.c2) # agrega calificación 2 / adds grade 2
67         temlis.append(self.c3) # agrega calificación 3 / adds grade 3
68
69         # 🔍 ordena calificaciones de mayor a menor (sin mover nombre) / sorts grades descending (keeps name fixed)
70         temlis = self.orden(temlis)
71
72         # 📌 agrega el registro ya ordenado a la lista principal / adds sorted record to main list
73         self.lista.append(temlis)
74
75         # ✅ mensaje de confirmación / confirmation message
76         messagebox.showinfo("Registro",f"registro guardado: {temlis}")
77
78     except ValueError:
79         # ⚠️ error si no es número / error if not a number
80         messagebox.showerror('Error', 'Introduce datos')
81
82     def orden(self,lis):
83         # 🔍 ordenamiento tipo burbuja para posiciones 1 a n (sin afectar el nombre en [0])
84
85         ext = len(lis)
86         for i in range(1, ext):
87             for j in range(1, ext - 1 - i):
88                 if lis[j] < lis[j+1]: # compara y cambia si el siguiente es mayor / compare and swap if next is larger
89                     aux = lis[j]
90                     lis[j] = lis[j+1]
91                     lis[j+1] = aux
92
93
94
95     if __name__ == "__main__":
96         app = Ventana() # 🖱️ instancia la clase y ejecuta la app / instantiates class and runs the app
97

```

Ilustración 52 Código Tarea 1

Registro de calificación

Nombre:

Calificación 1:

Calificación 2:

Calificación 3:

Agregar otro

Salir

Ilustración 53 Salida Tarea 1

Reaso 2 (preexamen)

Descripción: Programa que mediante una ventana solicita al usuario, 3 datos, cada uno debe cumplir una característica específica (que sean minúsculas, mayúsculas y números), y que al momento de presionar el botón de agregar se ingresen a la lisbox de la lateral, esto solo se podrá realizar si los datos ya fueron validados.

```

1  """
2  ^ hacer un programa que mediante una vetrana en tkinter realce lo sogueitne:
3  1 la ventana deve co tener 3 cjas de texto y 2 botones
4  2. el rrograma validara que la primer caja de texto solo permita letras minúsculas, en
5  la segunda caja solo letras mayúsculas, y en ña 3 cja de texto solo numeros, dicha validacion lo realizara
6  primer boton, si algun dato es incorrecto, mostrara un mensaje de error y se borrara la caja de texto.
7  3. únicamente cuando las 3 cajas de texto tengan informacion y esten validadas, se concatenaran
8  y se agregaran a una lista visual (lisbox) y en una etiqueta se mostrara el numero de elementos de la lista
9
10 (el que esta mal se borra al verificar)
11 """
12
13 from tkinter import *
14 from tkinter import messagebox
15 # from ValidarReaso import Validaciones # Importa la clase Validaciones desde otro archivo / imports the Validaciones class fr
16
17 class Principal():
18     def __init__(self): # solo se ejecuta una vez / runs only once
19         self.ventana = Tk() # Crea la ventana principal / creates the main window
20         self.ventana.geometry("550x250") # tamaño de ventana / sets window size
21         self.lista = [] # Lista para guardar los datos ingresados / list to store entered data
22         self.est = False
23         self.lista = []
24         self.contador = 0
25         self.conincidencias = 0
26         self.inicio()
27
28     def inicio(self):
29
30         Label(self.ventana, text= "Letras minusculas").place(x = 10, y = 20)
31         self.minus = Entry(self.ventana)
32         self.minus.place(x = 10, y = 50)
33         Label(self.ventana, text= "Letras mayúsculas").place(x = 150, y = 20)
34         self.mayus = Entry(self.ventana)
35         self.mayus.place(x = 150, y = 50)
36         Label(self.ventana, text= "Numeros").place(x = 300, y = 20)
37         self.num = Entry(self.ventana)
38         self.num.place(x = 300, y = 50)
39         self.listview = Listbox(self.ventana, width= 15, bg= "black", activestyle= "dotbox", fg="white")
40         # Listbox donde se muestran los valores agregados / Listbox showing added values
41         self.listview.place(x=450, y = 20)
42         self.prinCont = Label(self.ventana, text= "0")
43         self.prinCont.place(x = 150, y = 80)
44         Button(self.ventana, text= "Validar", command= self.validar).place(x = 120, y = 120)
45         Button(self.ventana, text= "Agregar", command= self.agregar).place(x = 200, y = 120)
46
47
48         self.ventana.mainloop()
49
50     def validar(self):
51         try:
52             min = self.minus.get()
53             may = self.mayus.get()
54             nu = self.num.get()
55             posicionInicio = 0
56

```

```

57     if self.validmin(min, 0) and self.validMay(may, 0) and self.validNum(nu):
58         self.est = True
59         messagebox.showinfo("Completo", "Datos validos")
60         return 0
61
62     self.est = False
63     messagebox.showerror("Error", "Dato no valido")
64
65 except ValueError:
66     messagebox.showerror("Campo bacio", "campo basio")
67
68 def agregar(self):
69     try:
70         if self.est :
71             registro = self.palMinus + self.palMayus + str(self.palNumero)
72             # registro.append(self.palMinus)
73             # registro.append(self.palMayus)
74             # registro.append(self.palNumero)
75             self.lista.append(registro)
76             self.listview.insert((self.listview.size() + 1),self.lista[self.contador])
77             # self.listview.insert(self.lista)
78             self.est = False
79             self.contador += 1
80             self_MINUS.delete(0, END)
81             self_MAYUS.delete(0, END)
82             self_NUM.delete(0,END)
83             self.printCont.config(text = str(self.contador))
84
85             self.printCont.config(text = str(self.contador))
86         else:
87             messagebox.showerror("Error", "Validacion incompleta")
88
89     except ValueError:
90         messagebox.showerror("Error", "Fallo el registro")
91
92
93
94 def validmin(self, letr, cnt):
95     if cnt < len(letr):
96         if ord(letr[cnt]) >= 97 and ord(letr[cnt]) <= ord("z"):
97             self.conConincidencias += 1
98
99         cnt += 1
100        return self.validmin(letr,cnt )
101
102    elif self.conConincidencias == len(letr):
103        self.palMinus = letr
104        self.conConincidencias = 0
105        return True
106
107    self.conConincidencias = 0
108    self_MINUS.delete(0,END)
109    return False

```

```

110
111     def validMay(self,letr, cnt):
112         if cnt < len(letr):
113             if ord(letr[cnt]) >= ord("A") and ord(letr[cnt]) <= ord("Z"):
114                 self.conConincidencias += 1
115
116             cnt += 1
117             return self.validMay(letr,cnt )
118
119         elif self.conConincidencias == len(letr):
120             self.palMayus = letr
121             self.conConincidencias = 0
122             return True
123
124         self.conConincidencias = 0
125         self.mayus.delete(0,END)
126         return False
127
128     def validNum(self, caracter):
129
130         try:
131             if caracter.isdigit():
132                 self.palNumero = caracter
133                 return True
134             else:
135                 a = 0.0
136                 a = float(caracter)
137                 self.palNumero = caracter
138
139
140         except ValueError:
141             self.num.delete(0,END)
142             return False
143
144
145
146
147
148     if __name__=="__main__":
149         app = Principal()

```

Ilustración 54 Código Repaso 2

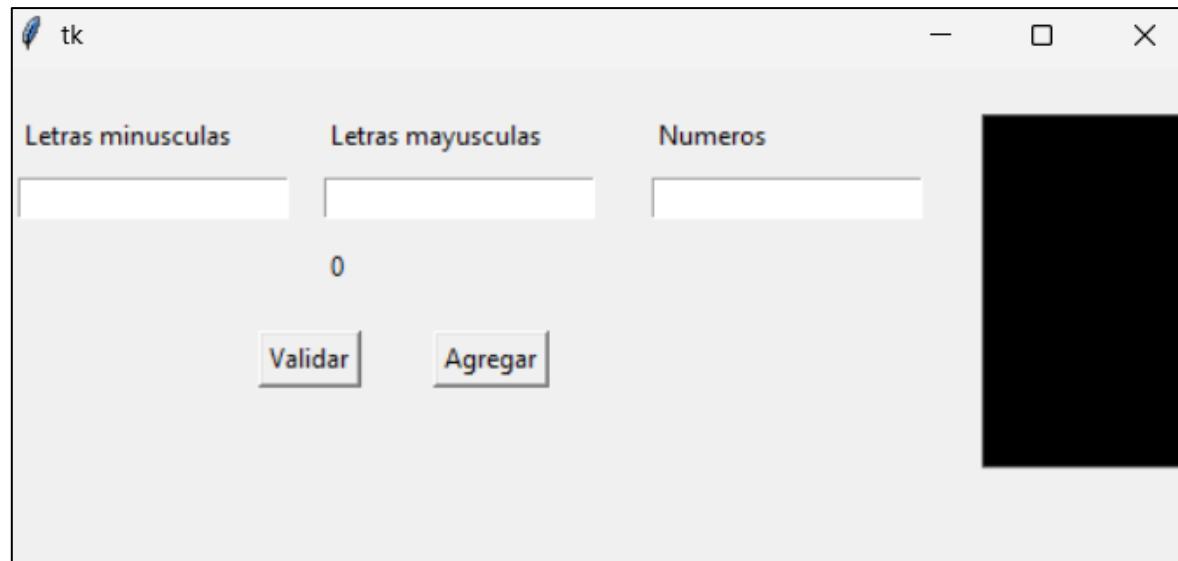


Ilustración 55 Salidas Repaso 2

Parcial 3:

Practica 1 (Ordenamiento):

Descripción: se desarrolla un programa que nos permite ingresar datos a una lista, para después ordenarlos ya sea de forma ascendente o descendente, para cada forma de ordenar se utiliza un algoritmo de ordenamiento diferente si es de menor a mayor se emplea el algoritmo de burbuja, mientras que de mayor a menor se utiliza el método de selección. Se implementan validaciones para garantizar que los numero ingresados sean solamente de dos dígitos, y que si sean números.

```

1 from tkinter import * # Importa todas las clases y funciones de Tkinter / Import all Tkinter classes and functions
2 from tkinter import messagebox # Importa messagebox para mostrar mensajes emergentes / Used for pop-up messages
3 from validaciones import validaciones1 # Importa la clase validaciones1 desde otro archivo / Import external validation class
4 import numpy as np # Importa NumPy para manejar arreglos numéricos / Imports NumPy for array manipulation
5
6 # Clase principal del programa / Main class of the program
7 class Principal():
8     def __init__(self):
9         # Configuración inicial de la ventana principal / Initial setup of the main window
10        self.ven = Tk() # Crea la ventana principal / Creates main window
11        self.val = validaciones1() # Crea un objeto de la clase validaciones1 / Creates an instance of validation class
12
13        # Define dimensiones y posición de la ventana / Defines window dimensions and position
14        ancho = 320 # Ancho de la ventana / Window width
15        alto = 250 # Alto de la ventana / Window height
16        ventana_alto = self.ven.winfo_screenwidth() # Obtiene ancho de pantalla / Get screen width
17        ventana_ancho = self.ven.winfo_screenheight() # Obtiene alto de pantalla / Get screen height
18        x = (ventana_alto // 2) - (ancho // 2) # Calcula coordenada X para centrar ventana / Calculate X coordinate
19        y = (ventana_ancho // 2) - (alto // 2) # Calcula coordenada Y para centrar ventana / Calculate Y coordinate
20        self.ven.geometry(f"[{ancho}x{alto}+{x + 500}+{y + 200}]") # Aplica tamaño y posición / Apply window size and position
21
22        # self.ven.geometry("320x210") # Define tamaño de la ventana / Set window size
23        # self.ven.configure(start_position = "center")
24
25        self.listaNum = [] # Lista donde se almacenarán números válidos / List to store valid numbers
26
27
28    def inicio(self):
29        # Entrada de texto donde el usuario escribe un dato / Text entry for user input
30        self.dato = Entry(self.ven)
31        self.dato.place(x = 30, y = 10) # Posiciona el cuadro de texto / Places the input field on window
32
33        # Configuración del modo de estructura de datos / Sets the data structure mode
34        self.modo = StringVar(value="Pilas") # valor predeterminado / Default value
35        Radiobutton(self.ven, text = "Pilas", variable= self.modo, value= "Pilas").place(x = 40, y = 40) # Modo pila / Stack mode
36        Radiobutton(self.ven, text = "Colas", variable= self.modo, value= "Colas").place(x = 90, y = 40) # Modo cola / Queue mode
37
38        # Configuración del tipo de ordenamiento / Sets sorting order
39        self.tipoDeOrden = StringVar(value = "Men_a_Mayor")
40        Radiobutton(self.ven, text = "Asendente", variable= self.tipoDeOrden, value= "Men_a_Mayor").place(x = 90, y = 150)
41        Radiobutton(self.ven, text = "Desendente", variable= self.tipoDeOrden, value= "Mayor_a_Men").place(x = 90, y = 150)
42
43        # Botones de acción / Action buttons
44        Button(self.ven, text= "Validar", command= self.validarCaja, width = 10).place(x = 100, y = 90) # Valida entrada / Validate input
45        Button(self.ven, text= "Eliminar", command= self.eliminarDatos, width = 10).place(x = 100, y = 120) # Elimina un dato / Delete data
46        self.label = Label(self.ven, text = "Número") # Etiqueta informativa / Informative label
47        self.label.place(x = 5, y = 70)
48        Button(self.ven, text= "Ordenar", command= self.ordenarDatos, width = 10).place(x = 100, y = 190) # Ordena los datos / Sort numbers
49
50
51    # Listbox para mostrar visualmente los números válidos / Listbox to visually show valid numbers
52    self.listViewValid = Listbox(
53        self.ven,
54        height = 10,
55        width = 10, # Ancho de la lista / Width
56        bg = "white", # Fondo blanco / White background
57        selectmode= "dotbox" # Estilo al seleccionar / Selection style
58    )

```

```

57     # fg= "white"      # ● Texto blanco / White text
58     font = ("Helvetica", 12)
59   )
60 self.listVisrual.place(x = 190, y = 10) # ♡ Posición del Listbox / Place list visually on window
61
62 self.ven.mainloop() # ☐ Mantiene la ventana abierta / Keeps the window running until closed
63
64
65 def ordenarDat(self):
66   # 📈 Método para ordenar los datos del Listbox / Method to sort data from Listbox
67   self.listaNum = list(self.listVisrual.get(0, END)) # 📁 Obtiene todos los elementos de la lista / Gets all list elements
68
69   if (len(self.listaNum) <= 0): # 🚫 Verifica que la lista no esté vacía / Check list not empty
70     messagebox.showerror("Error", "Lista vacía") # ▲ Mensaje de error si está vacía / Show error message
71     return 0
72
73   self.arreglo = np.array(self.listaNum) # ✖️ Convierte la lista en arreglo NumPy / Convert list to NumPy array
74
75   if self.tipoDeOrden.get() == "Menor a Mayor":
76     # 📈 Ordenamiento por método burbuja (ascendente) / Bubble sort (ascending)
77     for i in range(0, len(self.arreglo)):
78       for x in range(0, len(self.arreglo) - i - 1):
79         if (self.arreglo[x] > self.arreglo[x + 1]):
80           aux = self.arreglo[x]
81           self.arreglo[x] = self.arreglo[x + 1]
82           self.arreglo[x + 1] = aux
83
84     else:
85       # 📈 Ordenamiento por selección (descendente) / Selection sort (descending)
86     pos = 0
87     for i in range(0, len(self.arreglo)):
88       pos = i
89       aux = int(self.arreglo[i])
90
91       for x in range(i, len(self.arreglo)):
92         if aux < int(self.arreglo[x]):
93           aux = int(self.arreglo[x])
94           pos = x
95       self.arreglo[pos] = self.arreglo[i]
96       self.arreglo[i] = str(aux)
97
98     print(self.arreglo) # 📈 Imprime el arreglo ordenado / Print sorted array
99     self.listVisrual.delete(0, END) # ✎ Limpia el Listbox / Clear Listbox
100    for i in self.arreglo:
101      self.listVisrual.insert(self.listVisrual.size()+1, i) # 📈 Inserta los datos ordenados / Insert sorted data
102
103
104 def eliminarData(self):
105   # 🗑️ Método para eliminar un elemento según el modo / Delete element depending on mode
106   if self.listVisrual.size() <= 0:
107     messagebox.showerror("Error", "Lista vacía") # ▲ Mensaje de error si la lista está vacía / Show error message
108     return
109
110   if self.modo.get() == "Pilas":
111     # 📈 Pila: último que entra, primero que sale / Stack: last in, first out

```

```

111         self.listVisrual.delete(self.listVisrual.size()-1)
112     else:
113         # Cola: primero que entra, primero que sale / Queue: first in, first out
114         self.listVisrual.delete(0)
115     self.label.config(text = f"Elementos en la lista: {str(self.listVisrual.size())}" ) # Actualiza cantidad / Update counter
116
117
118     def validarCaja(self):
119         # Método para validar el contenido del campo de texto / Method to validate text field input
120         valor = self.dato.get() # Obtiene el texto escrito en el Entry / Get input value
121
122         # Validación numérica / Numeric validation
123         if self.val.validacionesNumeros(valor):
124             if self.val.validarEntradas(valor): # Si pasa la validación / If passes validation
125                 self.listVisrual.insert(END, valor) # Agrega el número a la lista / Insert number into list
126                 self.dato.delete(0, END) # Limpia campo / Clear field
127             else:
128                 messagebox.showerror("Error", "Solo se permiten dos dígitos") # Error: solo dos dígitos / Only two digits allowed
129                 self.dato.delete(0, END)
130         else:
131             messagebox.showerror("Error", "No son números") # Error: no numérico / Not a number
132             self.dato.delete(0, END)
133
134     self.label.config(text = f"Elementos en la lista: {str(self.listVisrual.size())}" ) # Muestra cantidad actual / Show element count
135
136
137     # Punto de entrada del programa / Entry point of the program
138 if __name__=="__main__":
139     app = Prinicipal() # Crea instancia de la clase principal / Create instance of main class
140     app.inicio() # Llama al método para iniciar la ventana / Call main method to start GUI
141

```

Ilustración 57 código práctica 1

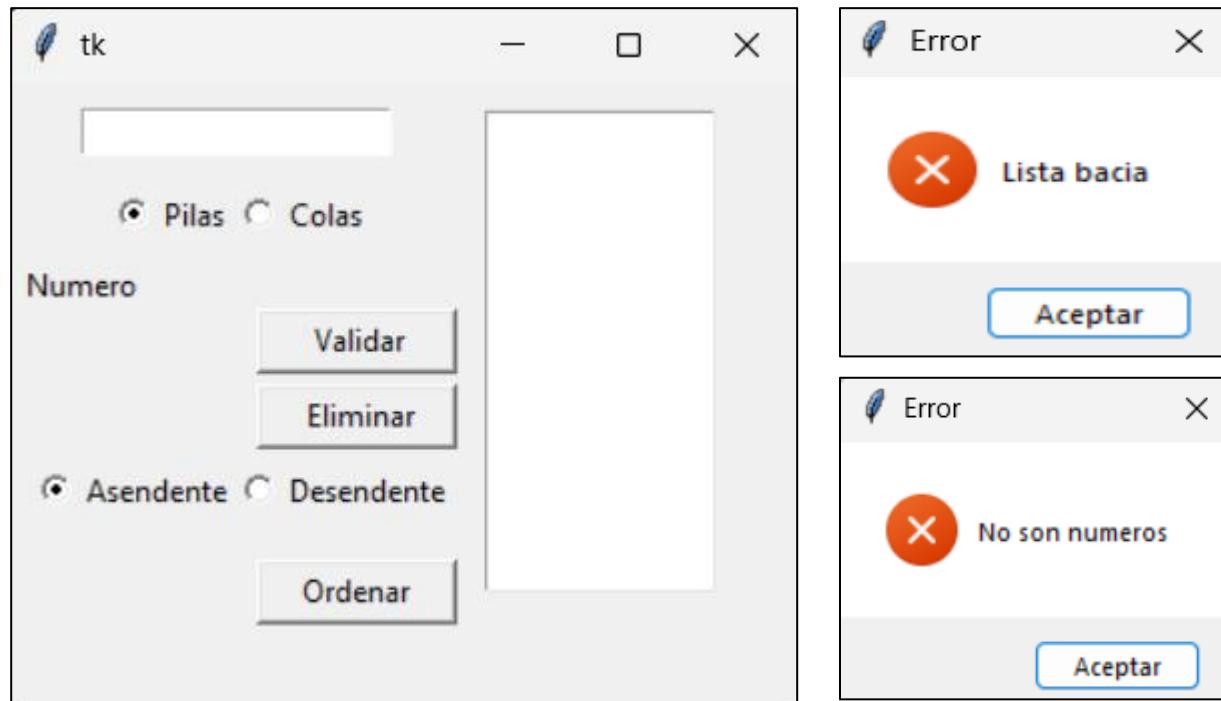


Ilustración 56 ventanas práctica 1

Practica 2 (Actualización de la practica 1 / Place-Holder):

Descripción : se nos enseño a implementar texto predefinido dentro de las cajas de texto, para ellos se necito implementar los eventos y una secuencia funciones como FocusIn, FocusOut.

```

1  # el programa con eventos en la caja de texto / Program with text box events
2  from tkinter import * # Importa todas las clases y funciones de Tkinter / Imports all classes and functions from Tkinter
3  from tkinter import messagebox # Permite mostrar mensajes emergentes / Allows pop-up messages
4  from validaciones import validaciones1 # Importa la clase validaciones1 desde otro archivo / Imports validation class from another file
5  import numpy as np # Biblioteca usada para cálculos numéricos / Library used for numerical computations
6
7  class Principal(): # Clase principal que controla la interfaz y la lógica / Main class controlling GUI and logic
8      def __init__(self):
9          self.val = validaciones1() # Crea un objeto de validaciones / Creates validation object
10         self.ven = Tk() # Crea la ventana principal de la aplicación / Creates main application window
11
12         ancho_ventana = 320 # Define ancho de la ventana / Window width
13         alto_ventana = 220 # Define alto de la ventana / Window height
14
15         # Obtener dimensiones de la pantalla / Get screen dimensions
16         ancho_pantalla = self.ven.winfo_screenwidth() # Ancho total de pantalla / Total screen width
17         alto_pantalla = self.ven.winfo_screenheight() # Alto total de pantalla / Total screen height
18
19         # Calcular posición para centrar / Calculate position to center
20         x = (ancho_pantalla // 2) - (ancho_ventana // 2) # Posición horizontal centrada / Horizontal centered position
21         y = (alto_pantalla // 2) - (alto_ventana // 2) # Posición vertical centrada / Vertical centered position
22
23         # Aplicar geometría / Apply geometry
24         self.ven.geometry(f'{ancho_ventana}x{alto_ventana}+{x}+{y}') # Define tamaño y posición de la ventana / Sets window size and position
25         self.lis = [] # Lista para almacenar elementos ingresados / List to store entered elements
26
27     def validarCaja(self, event):
28         valor = self.datos.get() # Obtiene texto de la caja de entrada / Gets text from entry field
29         if (self.val.ValidarLetra(valor)):
30             messagebox.showinfo("Correcto", "Si comienza con Mayusculas")
31         else:
32             messagebox.showinfo("Incorrecto", "No comienza con mayuscula")
33         if (self.val.ValidarNumeros(valor)):
34             messagebox.showinfo("Correcto", "Si es un numero")
35         else:
36             messagebox.showerror("Incorrecto", "No es un numero")
37
38         if (self.val.ValidacionesNumeros(valor)): # Comprueba si el valor contiene solo números / Checks if input contains only numbers
39             if (self.val.validarEntradas(valor)): # Verifica longitud válida (máx. 2 dígitos) / Validates correct input length (max 2 digits)
40                 self.lista.insert(self.lista.size()+1, valor) # Agrega el valor a la lista visual / Adds value to visual list
41                 self.datos.delete(0,END) # Limpia la caja de texto / Clears the text box
42             else:
43                 messagebox.showerror("Error", "Solo se permite 2 digitos") # Mensaje si supera longitud / Error if length exceeds limit
44                 self.datos.delete(0,END)
45             else:
46                 messagebox.showerror("Error", "No son numeros") # Mensaje si no es numérico / Shows error if input is not numeric
47                 self.datos.delete(0,END)
48
49         print(f'La cadena tiene {str(self.val.ValidarEntradas(valor))}')
50         self.label.config(text=f'Elementos en la lista: {str(self.lista.size())}') # Actualiza el conteo en la etiqueta / Updates label with e
51
52     def eliminarDatos(self):
53         if self.lista.size() <= 0: # Verifica si la lista está vacía / Checks if list is empty
54             messagebox.showerror("Error", "La lista esta vacia")
55             return
56         if self.modo.get() == 'Pilas': # Modo pila: último en entrar, primero en salir / Stack mode: last in, first out
57             # ultimo que entra primero que sale / last in, first out
58             self.lista.delete(self.lista.size()-1)
59         else: # Modo cola: primero en entrar, primero en salir / Queue mode: first in, first out
60             # primero que entra primero que sale / first in, first out
61             self.lista.delete(0)
62         self.label.config(text=f'Elementos en la lista: {str(self.lista.size())}') # Actualiza contador visual / Updates visual counter
63
64     def ordenar(self):
65         self.lis = list(self.lista.get(0,END)) # Convierte elementos del Listbox a una lista / Converts listbox items to Python list
66         if len(self.lis) <= 0:
67             messagebox.showerror("Error", "Lista vacia") # Error si no hay elementos / Error if list empty
68         else:
```

```

69     #burbuja / bubble sort
70     if self.modo2.get() == 'Burguja': # 🔍 Selecciona el método de burbuja / Selects bubble sort method
71         for i in range(0,len(self.lis)):
72             for x in range(0,len(self.lis)-1): # 🔍 Recorre lista varias veces / Loops through list multiple times
73                 if self.lis[x] > self.lis[x+1]: # 🔍 Compara elementos adyacentes / Compares adjacent elements
74                     aux = self.lis[x]
75                     self.lis[x] = self.lis[x+1]
76                     self.lis[x+1] = aux
77             print(self.lis) # 🔍 Muestra lista ordenada en consola / Displays sorted list
78             self.lista.delete(0,END) # 🔍 Limpia lista visual / Clears visual list
79             for i in self.lis:
80                 self.lista.insert(self.lista.size()+1, i) # 🔍 Inserta los elementos ordenados / Inserts sorted elements
81     else:
82         # seleccion / selection sort
83         p = 0 # 🔍 Índice del valor máximo / Index of maximum value
84         for i in range(0,len(self.lis)):
85             aux = int(self.lis[i]) # 🔍 Convierte elemento actual a entero / Converts current element to integer
86             p = i
87             for x in range(i,len(self.lis)): # 🔍 Recorre sublistas restante / Iterates remaining sublist
88                 # print(self.lis[x])
89                 if aux < int(self.lis[x]): # 🔍 Si encuentra un valor mayor, lo guarda / Stores larger value found
90                     aux = int(self.lis[x])
91                     p = x
92                     self.lis[p] = self.lis[i], # 🔍 Intercambia valores / Swaps values
93                     self.lis[i] = str(aux) # 🔍 Convierte de nuevo a cadena / Converts back to string
94             print(self.lis)
95             self.lista.delete(0,END) # 🔍 Limpia lista visual / Clears visual list
96             for i in self.lis:
97                 self.lista.insert(self.lista.size()+1, i) # 🔍 Inserta valores ordenados / Inserts sorted values
98
99     def quitar_placeholder(self, event):
100        if self.dato.get() == self.placeholder: # 🔍 Verifica si el texto es el placeholder / Checks if text is placeholder
101            self.dato.delete(0, END) # 🔍 Borra texto de guía / Deletes placeholder text
102            self.dato.config(fg="black") # 🔍 Cambia color de texto a negro / Changes text color to black
103
104    def poner_placeholder(self, event):
105        if self.dato.get() == "": # 🔍 Si la caja está vacía / If text box is empty
106            self.dato.insert(0, self.placeholder) # 🔍 Restaura texto guía / Restores placeholder text
107            self.dato.config(fg="gray") # 🔍 Cambia color del texto a gris / Changes text color to gray
108
109    def inicio(self):
110        # self.dato = Entry(self.ven)
111        # self.dato.place(x=50, y=10)
112        self.nombre = Entry(self.ven) # 🔍 Campo de texto auxiliar no usado / Auxiliary unused text field
113        self.nombre.place(x=1,y=1)
114        self.placeholder = "Escribe un número" # 🔍 Texto guía de la caja / Placeholder text
115        self.dato = Entry(self.ven, fg="gray") # 🔍 Caja de texto principal con color gris / Main text box with gray color
116        self.dato.insert(0, self.placeholder) # 🔍 Inserta texto guía inicial / Inserts initial placeholder
117        self.dato.bind("<FocusIn>", self.quitar_placeholder) # 🔍 Evento al hacer clic / Event when focus in
118        self.dato.bind("<FocusOut>", self.poner_placeholder) # 🔍 Evento al perder foco / Event when focus out
119        self.dato.bind("<Return>", self.validarCaja) # 🔍 Evento al presionar Enter / Event when pressing Enter
120        self.dato.place(x=50, y=10, width=100) # 🔍 Posición y tamaño de la caja / Sets text box position and size
121        self.modo = StringVar(value="Pilas") # 🔍 Variable que almacena el modo de eliminación / Stores removal mode
122        Radiobutton(self.ven, text="Pilas", variable=self.modo, value="Pilas").place(x=50,y=40) # 🔍 Botón para modo pila / Radio for stack mode
123        Radiobutton(self.ven, text="Colas", variable=self.modo, value="Colas").place(x=100,y=40) # 🔍 Botón para modo cola / Radio for queue mode
124        Button(self.ven, text="Validar", command=self.validarCaja, width=10).place(x=100,y=90) # 🔍 Botón para validar entrada / Button to validate
125        Button(self.ven, text="Eliminar", command=self.eliminarDatos, width=10).place(x=100,y=120) # 🔍 Botón para eliminar dato / Button to delete
126        self.modo2 = StringVar(value="Burguja") # 🔍 Variable que controla tipo de ordenamiento / Controls sorting method
127        Radiobutton(self.ven, text="Burguja", variable=self.modo2, value="Burguja").place(x=50,y=150) # 🔍 Opción burbuja / Bubble sort option
128        Radiobutton(self.ven, text="Selección", variable=self.modo2, value="Selección").place(x=100,y=150) # 🔍 Opción selección / Selection sort
129        Button(self.ven, text="Ordenar", command=self.ordenar, width=10).place(x=100,y=180) # 🔍 Botón para ordenar lista / Button to sort list
130        self.label = Label(text="Número") # 🔍 Etiqueta de estado / Status label
131        self.label.place(x=5,y=70)
132        self.lista = Listbox(self.ven, height=10, width=10, bg="white", font=("Helvetica", 12)) # 🔍 Lista visual para mostrar elementos / Visual
133        self.lista.place(x=190, y=10)
134        self.ven.mainloop() # 🔍 Inicia el bucle principal de la aplicación / Starts main event loop
135
136    if __name__=='__main__':
137        app = Principal() # 🔍 Crea la instancia principal de la clase / Creates main class instance
138        app.inicio() # 🔍 Ejecuta la interfaz gráfica / Runs graphical interface
139

```

Ilustración 58 Codigó practica 2

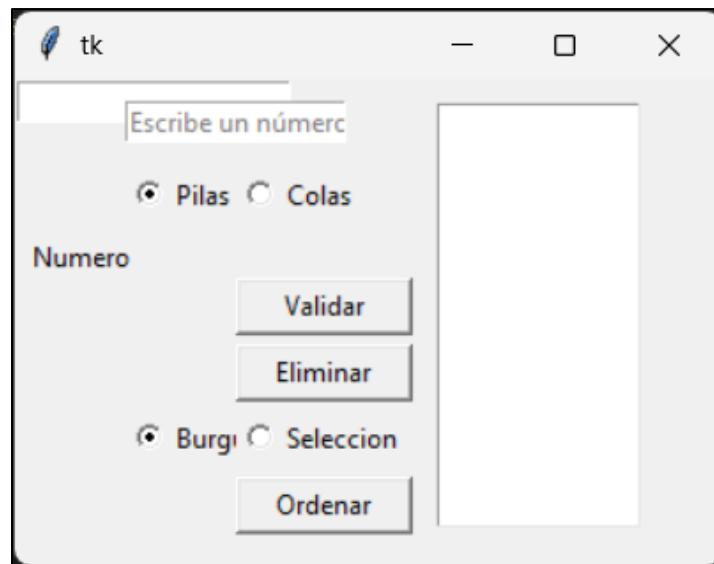


Ilustración 59 Ventana Práctica 2

Práctica 3:

Descripción: Se genera una clave que se guarda junto con los datos de nombre, sexo, teléfono y domicilio, lo cual se guarda en una list-box, antes de eso se validad que los datos sean validos (nombre: solo letras, teléfono: solo números).

```

1  from tkinter import * # Importa todas las clases y funciones de Tkinter / import all Tkinter classes and functions
2  from tkinter import messagebox # Importa messagebox para mostrar mensajes emergentes / Used for pop-up messages
3  from validaciones import validaciones1 # Importa la clase validaciones1 desde otro archivo / Import external validation class
4  import numpy as np # Importa NumPy para manejo de arreglos (aunque no se usa en este ejemplo) / Imports NumPy (not used here)
5
6  # Clase principal que gestiona toda la interfaz / Main class that manages the entire interface
7  class Principal():
8      def __init__(self):
9          # Creación de la ventana principal / Creation of the main window
10         self.ven = Tk() # Crea la ventana principal / Creates main window
11         self.ven.title("Práctica 3") # Título de la ventana / Sets window title
12
13         self.val = validaciones1() # Crea instancia de la clase de validaciones / Creates an instance of validation class
14
15         # Configuración de tamaño de la ventana / Set window dimensions
16         ancho_ventana = 350
17         alto_ventana = 250
18
19         # Obtiene dimensiones de pantalla / Get screen dimensions
20         ancho_pantalla = self.ven.winfo_screenwidth()
21         alto_pantalla = self.ven.winfo_screenheight()
22
23         # Cálculo para centrar ventana en pantalla / Calculate coordinates to center window
24         x = (ancho_pantalla // 2) - (ancho_ventana // 2)
25         y = (alto_pantalla // 2) - (alto_ventana // 2)
26
27         # Aplica posición y tamaño de la ventana / Apply window size and position
28         self.ven.geometry(f"{ancho_ventana}x{alto_ventana}+{x}+{y}")
29
30
31     def inicio(self):
32         # Cajas de texto con placeholders para guiar al usuario / Text boxes with placeholders to guide user input
33
34         # Caja de texto para el nombre / Text box for name
35         self.placeholderNom = "Nombre"

```

```

36     self.nombre = Entry(self.ven, fg="gray")
37     self.nombre.insert(0, self.placeholderNom)
38     self.nombre.bind("<FocusIn>", self.quitar_placeholderNom) # 🔍 Evento al enfocar / Event when focused
39     self.nombre.bind("<FocusOut>", self.poner_placeholder) # 🔍 Evento al salir / Event when unfocused
40     # self.nombre.bind("<Return>", self.validarCaja) # acción si presionas enter / Press enter to validate
41     self.nombre.place(x=10, y=10, width=100)
42
43     # 📞 Caja de texto para el teléfono / Text box for phone number
44     self.placeholderTel = "Teléfono"
45     self.telefono = Entry(self.ven, fg="gray")
46     self.telefono.insert(0, self.placeholderTel)
47     self.telefono.bind("<FocusIn>", self.quitar_placeholderTel)
48     self.telefono.bind("<FocusOut>", self.poner_placeholder)
49     # self.telefono.bind("<Return>", self.validarCaja)
50     self.telefono.place(x=120, y=10, width=100)
51
52     # 🏠 Caja de texto para domicilio / Text box for address
53     self.placeholderDom = "Domicilio"
54     self.domicilio = Entry(self.ven, fg="gray")
55     self.domicilio.insert(0, self.placeholderDom)
56     self.domicilio.bind("<FocusIn>", self.quitar_placeholderDom) # 🔍 Detecta si estás dentro / Detect if focused
57     self.domicilio.bind("<FocusOut>", self.poner_placeholder) # 🔍 Detecta si sales / Detect if unfocused
58     self.domicilio.bind("<Return>", self.validarCaja) # ⚡ Ejecuta validación al presionar Enter / Run validation on Enter
59     self.domicilio.place(x=250, y=10, width=100)
60
61     # 💇 Sección para seleccionar el sexo / Section to select gender
62     Label(self.ven, text = "Sexo: ").place(x = 10, y = 30)
63     self.modo = StringVar(value="F") # Valor predeterminado Femenino / Default value Female
64     Radiobutton(self.ven, text="F", variable=self.modo, value="F").place(x=10,y=50) # Opción Femenino / Female option
65     Radiobutton(self.ven, text="M", variable=self.modo, value="M").place(x=10,y=70) # Opción Masculino / Male option
66
67     # 📄 Listbox para mostrar los registros / Listbox to display registered people
68     self.lista = Listbox(self.ven, height=6, width=35, bg="white",font=("Helvetica", 12))
69     self.lista.place(x=10, y=100)
70
71     # 🖱 Botón para agregar registro / Button to add a record
72     Button(self.ven, text = "Agregar", command= self.validarCaja).place(x= 210, y = 60, width = 100, height = 20)
73
74     self.ven.mainloop() # 🕹 Mantiene la ventana abierta / Keeps the window running
75
76
77 def agregar(self):
78     # 🔍 Método reservado para futuras funciones de agregar / Placeholder for future add function
79     pass
80
81 def validarCaja(self, event = 0):
82     # 📋 Validación de campos de entrada / Validation of text box data
83     if (self.nombre.get() == self.placeholderNom
84         or self.telefono.get() == self.placeholderTel
85         or self.domicilio.get() == self.placeholderDom
86         or self.domicilio.get() == ""):
87
88         messagebox.showerror("Error", "Campo vacíos") # 🔞 Error si hay campos vacíos / Error if any field is empty
89
90     else:
91         # messagebox.showinfo("Ventana", "👋Hola mundo👋")
92         nombre = self.nombre.get() # 📄 Obtiene nombre / Get name
93         telefono = self.telefono.get() # 📞 Obtiene teléfono / Get phone
94         domicilio = self.domicilio.get() # 🏠 Obtiene domicilio / Get address
95
96         # ✅ Validaciones: letras para nombre y números para teléfono / Validations: letters for name, numbers for phone
97         if self.val.ValidarLetra(nombre) and self.val.ValidacionesNumeros(telefono):
98             if self.modo.get() == "F":
99                 sexo = "Femenino"
100            else:
101                sexo = "Masculino"

```

```

102
103     # 🖼 Genera una clave combinando datos / Generate a key combining parts of input
104     clave = nombre[0] + telefono[0] + domicilio[2:]
105     persona = clave + " - " + nombre + " - " + telefono + " - " + domicilio + " - " + sexo
106
107     # 📑 Inserta registro en la lista / Insert record into list
108     self.lista.insert(END, persona)
109
110     # ✎ Limpia campos de texto / Clear input fields
111     self.nombre.delete(0, END)
112     self.telefono.delete(0, END)
113     self.domicilio.delete(0, END)
114
115     # ❌ Si el nombre es válido pero el teléfono no / If name is valid but phone invalid
116     elif self.val.ValidarLetra(nombre):
117         messagebox.showerror("Error", "Teléfono no valido")
118         self.telefono.delete(0, END)
119
120     # ❌ Si el teléfono es válido pero el nombre no / If phone is valid but name invalid
121     elif self.val.ValidacionesNumeros(telefono):
122         messagebox.showerror("Error", "Nombre no valido")
123         self.nombre.delete(0, END)
124
125
126     # 🔍 Métodos para gestionar placeholders de entrada / Methods to handle input placeholders
127
128     def quitar_placeholderNom(self, event):
129         # 🔍 Elimina el texto de ayuda cuando el usuario hace clic / Removes hint text when user clicks
130         if self.nombre.get() == self.placeholderNom:
131             self.nombre.delete(0, END)
132             self.nombre.config(fg="black")
133             return 0
134
135
136     def quitar_placeholderTel(self, event):
137         if self.telefono.get() == self.placeholderTel:
138             self.telefono.delete(0, END)
139             self.telefono.config(fg="black")
140             return 0
141
142     def quitar_placeholderDom(self, event):
143         if self.domicilio.get() == self.placeholderDom:
144             self.domicilio.delete(0, END)
145             self.domicilio.config(fg="black")
146             return 0
147
148     def poner_placeholder(self, event):
149         # 🔍 Restaura texto de ayuda si el campo queda vacío / Restore hint text if field left empty
150         if self.nombre.get() == "":
151             self.nombre.insert(0, self.placeholderNom)
152             self.nombre.config(fg="gray")
153         elif self.telefono.get() == "":
154             self.telefono.insert(0, self.placeholderTel)
155             self.telefono.config(fg="gray")
156         elif self.domicilio.get() == "":
157             self.domicilio.insert(0, self.placeholderDom)
158             self.domicilio.config(fg="gray")
159
160
161     # 🚀 Punto de entrada principal / Main entry point
162     if __name__ == "__main__":
163         app = Principal() # ✨ Crea una instancia de la clase / Create an instance of the class
164         app.inicio() # 🖥 Inicia la interfaz gráfica / Start the graphical interface
165

```

Ilustración 60 Código Práctica 3

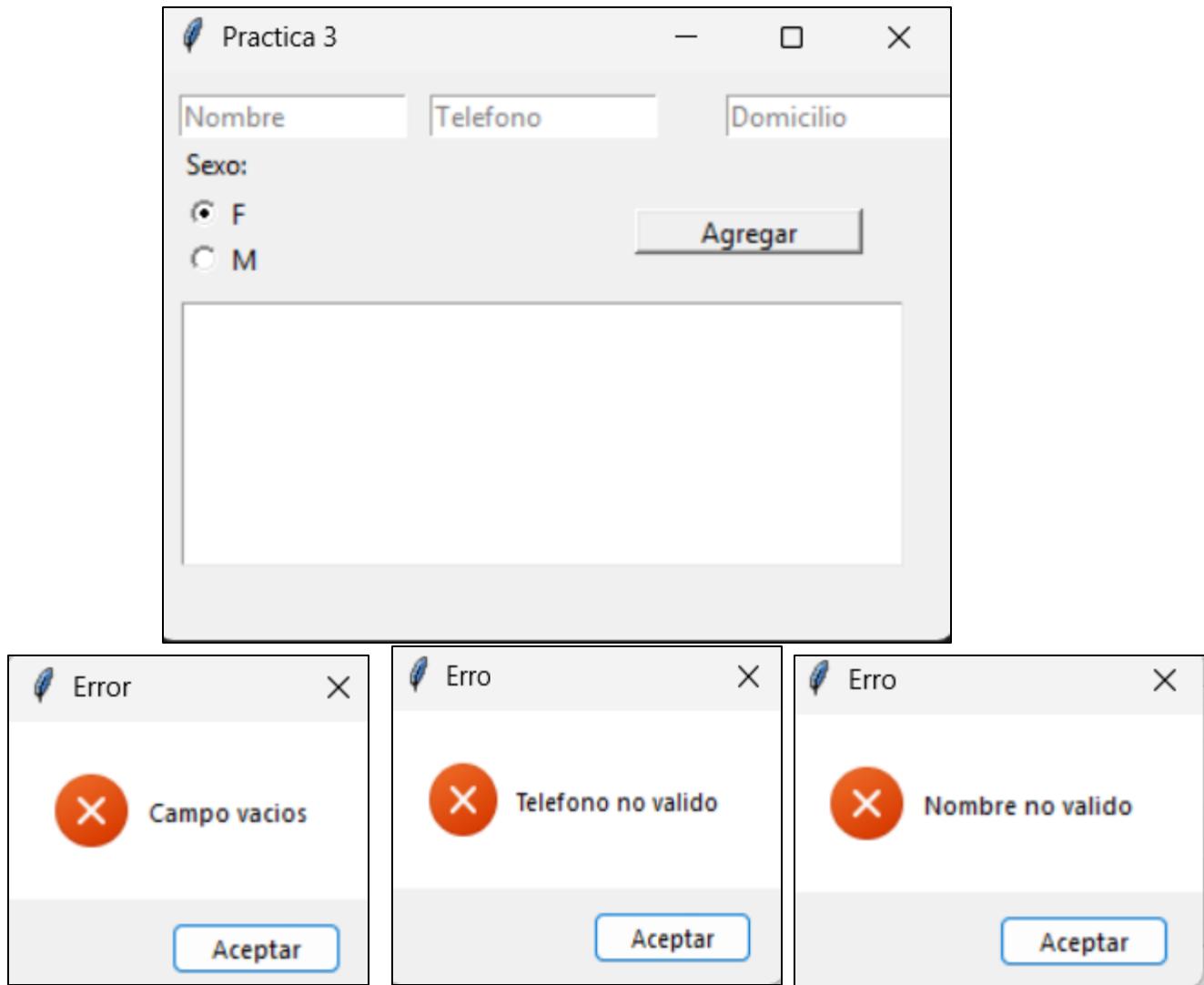


Ilustración 61 Ventanas Practica 3

Practica 4(Table):

Descripción: Primera práctica en la cual se implementan las tablas en Python, se utiliza en esta ocasión para guardar un registro de usuario, con una clave única autogenerada, complementada con el nombre del usuario.

```

1  from tkinter import * # Importa todas las clases y funciones de Tkinter / Import all Tkinter classes and functions
2  from tkinter import messagebox # Importa messagebox para mostrar mensajes emergentes / Used for pop-up messages
3  from tkinter import ttk # Importa ttk para usar widgets avanzados (Treeview, Scrollbar, etc.) / Imports ttk for advanced widgets
4  from validaciones import validaciones1 # Importa la clase validaciones1 desde otro archivo / Import external validation class
5  import numpy as np # Importa NumPy para operaciones numéricas / Imports NumPy for numeric operations
6  import random # Importa random para generar números aleatorios / Imports random for generating random numbers
7
8  class Principal(): # Clase principal del programa / Main class of the program
9      def __init__(self):
10          self.ven = Tk() # Crea la ventana principal / Creates main window
11          self.ven.title("Practica 3") # Título de la ventana / Sets window title
12          self.val = validaciones1() # Crea un objeto de la clase validaciones1 / Creates an instance of validation class
13          ancho_ventana = 500 # Ancho de la ventana / Window width
14          alto_ventana = 300 # Alto de la ventana / Window height
15
16          # Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
17          # Obtener dimensiones de la pantalla
18          ancho_pantalla = self.ven.winfo_screenwidth()
19          alto_pantalla = self.ven.winfo_screenheight()
20
21          # Calcular posición para centrar
22          x = (ancho_pantalla // 2) - (ancho_ventana // 2)
23          y = (alto_pantalla // 2) - (alto_ventana // 2)
24
25          # Aplicar geometría
26          self.ven.geometry(f"{ancho_ventana}x{alto_ventana}+{x}+{y}")
27
28          self.cont = 0 # Contador para generar claves únicas / Counter for unique IDs
29          self.bandera = False # Indica si se está en modo edición / Flag to indicate edit mode
30          self.renglon = -1 # Guarda el índice del renglón seleccionado / Stores selected row index
31          self.index = "" # Guarda parte de la clave para edición / Stores partial key for editing
32
33      def inicio(self):
34          #Caja de texto nombre:
35          # Campo para capturar el nombre del usuario / Text field to enter user's name
36          Label(self.ven, text= "Nombre").place(x = 10, y = 10)
37          self.nombre = Entry(self.ven, fg="blue")
38          self.nombre.place(x=10, y=30, width=100)
39
40          #caja de texto edad:
41          # Campo para capturar la edad / Text field to enter age
42          Label(self.ven, text= "Edad").place(x = 120, y = 10)
43          self.edad = Entry(self.ven, fg="green")
44          self.edad.place(x=120, y=30, width=100)
45
46          #Caja de texto domicilio:
47          # Campo para capturar el correo electrónico / Text field to enter email
48          Label(self.ven, text= "Correo").place(x = 250, y = 10)
49          self.correo = Entry(self.ven, fg="purple")
50          self.correo.place(x=250, y=30, width=100)
51
52          # Botones principales del programa / Main control buttons
53          Button(self.ven, text = "Añadir", command= self.agregarElemento).place(x= 380, y = 50, width = 100, height = 30)
54          Button(self.ven, text = "Eliminar", command= self.eliminar).place(x= 380, y = 90, width = 100, height = 30)
55          Button(self.ven, text = "Seleccionar", command= self.seleccionar).place(x= 380, y = 130, width = 100, height = 30)
56
57          # dataGrid:
58          # Crea la tabla (Treeview) con encabezados / Creates data table (Treeview) with headers...
59          self.tabla = ttk.Treeview(self.ven, columns = columnas, show= "headings")
60          self.tabla.place(x = 10, y = 100, width = 350, height = 190)
61
62          # Configura encabezados y columnas centradas / Sets up headers and centers columns
63          for col in columnas:
64              self.tabla.heading(col, text = col)
65              self.tabla.column(col, anchor="center", width = 30)
66
67          # Barras de desplazamiento vertical y horizontal / Vertical and horizontal scrollbars
68          scrollly = ttk.Scrollbar(self.ven, orient = "vertical", command = self.tabla.yview)
69

```

```

70 scrollx=ttk.Scrollbar(self.ven, orient = "horizontal", command = self.tabla.xview)
71 scrollx.place(x = 360, y = 100, height = 190)
72 scrollx.place(x = 10, y = 280, width = 350 )
73
74 # ☐ Inicia el bucle principal de la interfaz / Starts main event loop
75 self.ven.mainloop()
76
77 def seleccionar(self):
78     # ☐ Obtiene la fila seleccionada en la tabla / Gets selected row in table
79     self.renglon = self.tabla.selection()
80
81 if self.renglon:
82     valores = self.tabla.item(self.renglon, "values") # ☠ Extrae los valores del renglón / Extracts row values
83     # print(valores)
84     self.index = valores[0] # ↗ Guarda la clave / Stores key value
85     self.index = self.index[:len(self.index)-2] # ☠ Elimina los últimos dos caracteres / Removes last two characters
86     # ☡ Inserta los datos en las cajas de texto / Inserts data back into entry fields
87     self.nombre.insert(0, valores[1])
88     self.edad.insert(0, valores[3])
89     self.correo.insert(0, valores[2])
90     self.bandera = True # ► Activa modo edición / Enables edit mode
91     return 0
92
93 messagebox.showerror("Error", "Elije un fila") # ▲ Mensaje si no hay selección / Error if no row is selected
94
95 def agregarElemento(self):
96     # ☐ Verifica si los campos están vacíos / Checks if any input fields are empty
97     if len(self.nombre.get()) == 0 or len(self.edad.get()) == 0 or len(self.correo.get()) == 0:
98         messagebox.showerror("Error", "Campos bacios") # ▲ Muestra error si hay campos vacíos / Shows error if empty fields
99     else:
100         # if self.val.validarLetrasRecursivo(self.nombre.get()):
101         #     print("Eureka funciono")
102
103     nombre = self.nombre.get() # ↗ Captura nombre / Gets name
104     edad = self.edad.get() # ☠ Captura edad / Gets age
105     correo = self.correo.get() # ☡ Captura correo / Gets email
106
107     if not(self.bandera): # ↗ Si no está en modo edición / If not in edit mode
108         self.cont += 1 # ☠ Incrementa el contador / Increases counter
109         # ↗ Genera clave única usando contador, número aleatorio y primeras letras del nombre / Generates unique key
110         clave = str(self.cont) + str(random.randint(1, 100)) + self.nombre.get()[0:2].upper()
111         self.tabla.insert("", "end", values = (clave, nombre, correo, edad)) # ☡ Inserta fila en la tabla / Inserts row in table
112
113     else:
114         # ↗ Modo edición activado / Edit mode activated
115         claveEd = self.index + nombre[:2].upper()
116         print("Modo edición Activado") # ☠ Mensaje en consola / Console message
117         # ☡ Actualiza los valores de la fila seleccionada / Updates selected row data
118         self.tabla.item(self.renglon, values = (claveEd, nombre, correo, edad))
119         self.bandera = False # ► Desactiva modo edición / Turns off edit mode
120         self.renglon = -1 # ☠ Reinicia indice / Resets row index
121         messagebox.showinfo("Correcto", "Datos Actualizados") # ☑ Mensaje de éxito / Success message
122
123     # ↗ Limpia los campos de texto / Clears all entry fields
124     self.nombre.delete(0, END)
125     self.edad.delete(0, END)
126     self.correo.delete(0, END)
127
128 def eliminar(self):
129     # ☠ Elimina la fila seleccionada de la tabla / Deletes selected row from table
130     self.renglon = self.tabla.selection()
131     if self.renglon:
132         self.tabla.delete(self.renglon) # ☠ Borra la fila / Deletes the row
133         messagebox.showinfo("Correcto", "Correcto, dato eliminado") # ☑ Confirmación / Success message
134         return 0
135     messagebox.showerror("Error", "Elije un fila") # ▲ Error si no hay selección / Error if no row selected
136
137
138
139 if __name__ == "__main__": # ☠ Punto de entrada del programa / Program entry point
140     app = Principal() # ☡ Crea instancia de la clase principal / Creates main class instance
141     app.inicio() # ☡ Llama al método para iniciar la interfaz / Calls method to start interface
142

```

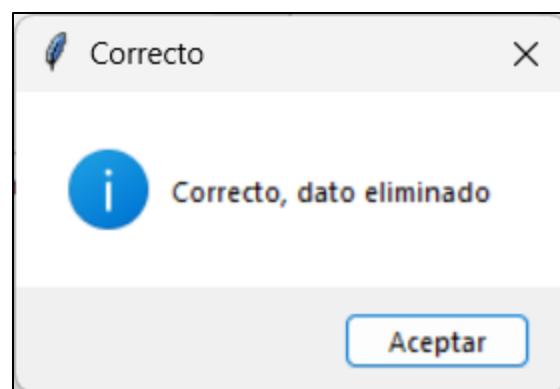
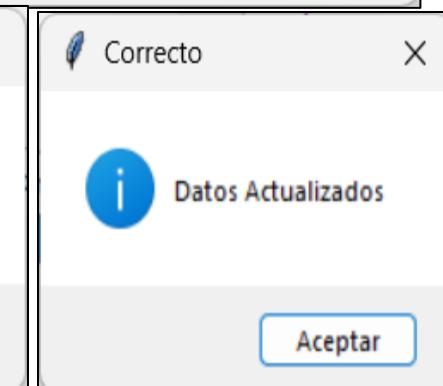
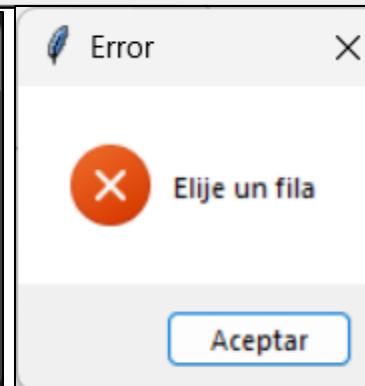
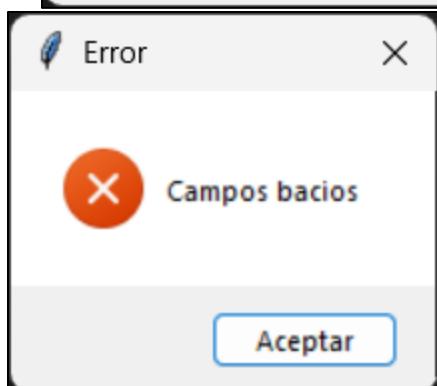
Ilustración 62 Código Práctica 4

Practica 3

Nombre	Edad	Correo
asd	12	asd@asd

Clave	Nombre	Correo	Edad
199AS	asd	asd@asd	12

Agregar
Eliminar
Seleccionar



Reaso 1(Tarea 1: Generar RFC):

Descripción: La practica consiste en realizar un programa que permita generar el RFC de cualquier persona, fue diseño libre, tuvimos la labor de implementar validaciones para garantizar que los datos ingresados fueran validos.

```

1  ^
2  ^ hacer unprograma que lea nombre, apellido paterno, y apellido materno
3  en 3 cajas separadas, ademas, leer, dia, mes y años de nacimiento en 3 cajas de texto
4  separadas.
5
6  al preciopnar un bton se agregara al un lis box, el rfc de la persona
7  ademas, contendra dos botones, para eliminar elemtnos del lisbox mediante pilas o colas
8
9  rfc =
10 primera letra y priera vocal del primer apellido
11 primera letra del segundo apellido
12 ultimos dos digititos del año
13 dos digotos del mes
14 dos digotos del dia
15 ...
16 from tkinter import * # Importa todas las clases y funciones de Tkinter / Import all Tkinter classes and functions
17 from tkinter import messagebox # Importa messagebox para mostrar mensajes emergentes / Used for pop-up messages
18 from validaciones import validaciones1 # Importa la clase validaciones1 desde otro archivo / Import external validation class
19
20 class Principal(): # Clase principal de la aplicacion / Main class of the application
21     def __init__(self):
22         self.ventana = Tk() # Crea la ventana principal / Creates the main window
23         self.ventana.geometry("450x350") # Define tamano de la ventana / Sets window size
24         self.valida = validaciones1() # Crea objeto de validacion / Creates validation object
25
26     def inicio(self):
27         # Campos para ingresar nombre y apellidos / Entry fields for name and surnames
28         Label(self.ventana, text = "Nombre").place(x = 3, y = 10)
29         self.name = Entry(self.ventana)
30         self.name.place(x = 3, y = 50, width= 80)
31
32         Label(self.ventana, text = "Apellido Paterno").place(x = 100, y = 10)
33         self.apPaterno = Entry(self.ventana)
34         self.apPaterno.place(x = 100, y = 50, width= 80)
35
36         Label(self.ventana, text = "Apellido Materno").place(x = 200, y = 10)
37         self.apMaterno = Entry(self.ventana)
38         self.apMaterno.place(x = 200, y = 50, width= 80)
39
40         # Campos para fecha de nacimiento / Fields for birth date
41         Label(self.ventana, text = "Dia").place(x = 3, y = 100)
42         self.dia = Entry(self.ventana)
43         self.dia.place(x = 3, y = 150, width= 80)
44
45         Label(self.ventana, text = "Mes").place(x = 100, y = 100)
46         self.mes = Entry(self.ventana)
47         self.mes.place(x = 100, y = 150, width= 80)
48
49         Label(self.ventana, text = "Año").place(x = 200, y = 100)
50         self.año = Entry(self.ventana)
51         self.año.place(x = 200, y = 150, width= 80)
52
53         # Botn para agregar y botones de seleccin de modo / Add button and mode selectors
54         Button(self.ventana, text = "Agregar", command= self.agregar).place(x = 10, y = 190)
55         self.modo = StringVar(value="Pilas") # valor predeterminado / default value
56
57         Radiobutton(self.ventana, text = "Pilas", variable= self.modo, value= "Pilas").place(x = 70, y = 195)
58         Radiobutton(self.ventana, text = "Colas", variable= self.modo, value= "Colas").place(x = 70, y = 220)
59
60         Button(self.ventana, text = "Eliminar", command= self.eliminar).place(x = 10, y = 220)
61
62         # Listbox para mostrar los RFC generados / Listbox to display generated RFCS
63         self.listvisual = Listbox(
64             self.ventana,
65             height = 10,          # Altura de la lista / List height
66             width = 16,           # Ancho de la lista / List width
67             bg = "white",         # Fondo blanco / White background
68             font = ("Helvetica", 12) # Tip y tamao de letra / Font type and size

```

```

69      )
70      self.listVisual.place(x = 300, y = 10) # 🕹️ Posición del Listbox / Position of the Listbox
71
72      # 🚧 Inicia el ciclo principal de la interfaz / Starts main loop
73      self.ventana.mainloop()
74
75  def eliminar(self):
76      # 🗑️ Elimina elementos del Listbox según el modo seleccionado / Deletes items using selected mode (stack or queue)
77
78  if self.listVisual.size() <= 0:
79      messagebox.showerror("Error", "lita vacía") # 🔞 Mensaje si la lista está vacía / Error if list is empty
80      return
81
82  if self.modo.get() == "Pilas":
83      # 🚧 Último que entra, primero que sale / Last In, First Out (stack)
84      self.listvisual.delete(self.listVisual.size()-1)
85  else:
86      # 🚧 Primero que entra, primero que sale / First In, First Out (queue)
87      self.listVisual.delete(0)
88
89
90  def agregar(self):
91      # ✨ Genera el RFC y lo agrega al Listbox / Generates RFC and adds it to the Listbox
92  try:
93      nombre = self.name.get().upper() # 📝 Convierte nombre a mayúsculas / Converts name to uppercase
94      primerApe = self.apPaterno.get().upper() # 📝 Apellido paterno en mayúsculas / Paternal surname uppercase
95      segundoApe = self.apMaterno.get().upper() # 📝 Apellido materno en mayúsculas / Maternal surname uppercase
96      año = self.año.get() # 📆 Año de nacimiento / Birth year
97      mes = self.mes.get() # 📆 Mes de nacimiento / Birth month
98      dia = self.dia.get() # 📆 Día de nacimiento / Birth day
99      est = True # ✅ Variable de control de validez / Validation flag
100
101     # ✖️ Validaciones de campos de texto / Validations for text inputs

```

```

102    if len(nombre) == 0 or not(self.valida.ValidarLetra(nombre)):
103        self.name.delete(0, END)
104        est = False
105
106    if len(primerApe) == 0 or not(self.valida.ValidarLetra(primerApe)):
107        self.apPaterno.delete(0, END)
108        est = False
109
110    if len(segundoApe) == 0 or not(self.valida.ValidarLetra(segundoApe)):
111        self.apMaterno.delete(0, END)
112        est = False
113
114    # ✅ Validación del año / Year validation
115    if len(año) != 4 or not(self.valida.ValidacionesNumeros(año)):
116        self.año.delete(0, END)
117        est = False
118
119    elif int(año) > 2025 or int(año) < 1900:
120        self.año.delete(0, END)
121        est = False
122
123    # ✅ Validación del mes / Month validation
124    if self.valida.validaMes(mes):
125        if len(mes) == 1:
126            mes = f"0{mes}" # ✨ Agrega cero inicial / Adds leading zero
127        else:
128            self.mes.delete(0,END)
129            est = False
130
131    # ✅ Validación del día / Day validation
132    if self.valida.validaDia(dia, mes):
133        if len(dia) == 1:
134            dia = f"0{dia}" # ✨ Agrega cero inicial / Adds leading zero

```

```

135     else:
136         self.dia.delete(0, END)
137         est = False
138
139         # Si todos los datos son válidos / If all data is valid
140         if est:
141             # Construye el RFC siguiendo la regla / Builds the RFC using the rule
142             rfc = f"{self.valida.extraerdatos(primerApe)}{segundoApe[0]}{nombre[0]}{año[2:]}/{mes}/{dia}"
143             self.listVisual.insert(END, rfc.upper()) # Agrega RFC al Listbox / Inserts RFC into the Listbox
144
145             # Limpia las cajas de texto / Clears all input fields
146             self.name.delete(0, END)
147             self.apPaterno.delete(0, END)
148             self.apMaterno.delete(0, END)
149             self.año.delete(0, END)
150             self.mes.delete(0, END)
151             self.dia.delete(0, END)
152
153
154         # Mensaje si los datos no son válidos / Error if data invalid
155         messagebox.showerror("Error", "Datos no validos")
156
157     except ValueError:
158         # Error al detectar campo vacío o formato incorrecto / Error for empty or invalid input
159         messagebox.showerror("Error", "Campo vacio")
160
161
162 if __name__ == "__main__": # Punto de inicio del programa / Program entry point
163     app = Principal() # Crea instancia de la clase principal / Creates instance of the main class
164     app.inicio() # Llama al método inicio / Calls the start method
165

```

Ilustración 64 Código Repaso 1

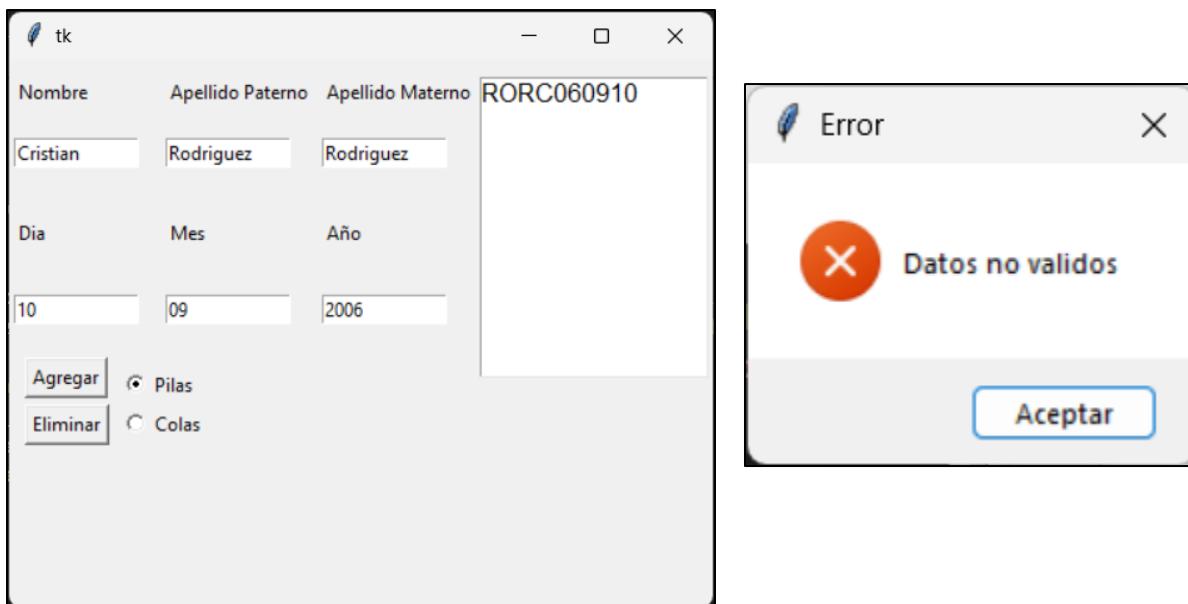


Ilustración 63 Ventanas Repaso 1

Validaciones:

Descripción: archivo que contiene una recopilación de validaciones diferentes métodos, que nos permite validar si son números, si son letras, si un día es valido e incluso si un mes es válido.

```

1  class Validaciones1(): # 🔴 Clase que contiene distintos métodos de validación / Class that holds multiple validation methods
2      def __init__(self):
3          self.con = 0 # 🟢 Contador interno usado en validaciones recursivas / Internal counter for recursive checks
4
5      def ValidacionesNumeros(self, num):
6          # 🔴 Verifica si toda la cadena está compuesta solo por números / Checks if entire string is digits only
7          if num.isdigit():
8              return True # ✅ Devuelve True si son solo números / Returns True if only digits
9          return False # ❌ De lo contrario devuelve False / Otherwise returns False
10
11     def ValidarLetra(self, dato):
12         c = 0 # 🟢 Contador auxiliar / Helper counter
13         dato = dato.upper() # 🟢 Convierte todo a mayúsculas para simplificar validación / Converts all to uppercase
14
15         for i in dato: # 🟤 Recorre cada carácter del texto / Loops through each character
16             if not(ord(i) >= 65 and ord(i) <= 90 or ord(i) == 32): # 🟢 Sólo se permiten letras (A-Z) y espacios / Only letters and spaces allowed
17                 return False # ❌ Devuelve False si encuentra un carácter no permitido / Returns False if invalid char
18             return True # ✅ Si pasa todas las comprobaciones, es válido / Returns True if all chars are valid
19
20     def validarLetrasRecurSivo(self, dato, con = 0):
21         # 🟤 Versión recursiva para validar letras / Recursive version for letter validation
22         if con >= len(dato): # 🟢 Caso base: si ya recorrió toda la cadena / Base case: end of string
23             return True
24
25         if ord(dato[con].upper()) >= 65 and ord(dato[con].upper()) <= 90 or ord(dato[con].upper()) == 32:
26             con += 1 # 🟤 Avanza al siguiente carácter / Move to next character
27             return self.validarLetrasRecurSivo(dato, con) # 🟤 Llamada recursiva / Recursive call
28
29         return False # ❌ Si encuentra un carácter no válido, retorna False / Invalid char found
30
31     def validarEntradas(self, dato):
32         # 🔴 Verifica que la longitud del dato sea exactamente 2 / Ensures input has exactly two characters
33         if len(dato) == 2:
34             return True # ✅ Válido si tiene dos caracteres / Valid if two characters
35         return False # ❌ De lo contrario, inválido / Otherwise invalid
36
37     def extraerDatos(self, dato):
38         # ❌ Extrae una combinación de letras (por ejemplo, para generar iniciales) / Extracts letters (e.g., initials)
39         c = 0
40         dato = dato.upper() # 🟢 Convierte el texto a mayúsculas / Converts text to uppercase
41         dat = "" # 🟢 Variable donde se guardará el resultado / Stores extracted result
42
43         if " " in dato: # 🔴 Si hay espacios, toma la última palabra / If there are spaces, take the last word
44             tem = dato.split(" ")
45             dato = tem[len(tem)-1]
46
47             ctem = 0 # 🟢 Contador temporal / temporary counter
48             for i in dato:
49                 if c >= 1:
50                     if i in ["A", "E", "I", "O", "U"]:
51                         dat += i
52                         c += 1
53                     elif ctem == len(dato) - 1: # 🔴 Si no hay vocal, usa la segunda letra / If no vowel found, use second letter
54                         dat += dato[1]
55                 else:
56                     dat += i # 🟤 Agrega la primera letra / Adds first letter
57                     c += 1
58             if c >= 2:
59                 break # 🔴 Detiene cuando ya tiene dos letras / Stops after two letters
60             ctem += 1
61             return dat # 🟤 Devuelve el resultado / Returns result
62
63     def validaMes(self, mes):
64         # 🟢 Valida que el mes esté entre 1 y 12 / Validates month range
65         if len(mes) <= 2: # 🟢 Debe tener máximo 2 dígitos / Must have max 2 digits
66             if mes.isdigit(): # 🔴 Debe ser numérico / Must be numeric
67                 if int(mes) > 0 and int(mes) <= 12: # 🟤 Rango válido de meses / Valid month range
68                     return True
69             return False # ❌ Mes no válido / Invalid month

```

```

71 def validaDia(self, dia, mes = -1):
72     # 📌 Verifica que el día sea válido según el mes / Validates day according to month
73     try:
74         dia_int = int(dia) # 📌 Convierte a entero / Converts to integer
75         mes_int = int(mes)
76
77         if mes_int >= 1 and mes_int <= 12: # ✅ Verifica que el mes sea válido / Checks valid month
78             if mes_int in [2,4,6,9,11]: # 📌 Meses con 30 o menos días / Months with ≤ 30 days
79                 if mes_int == 2 and dia_int <= 28: # 📌 Febrero máximo 28 / February up to 28
80                     return True
81                 elif dia_int <= 30: # 📌 Abril, junio, septiembre, noviembre hasta 30 / Up to 30
82                     return True
83             else:
84                 if dia_int <= 31: # 📌 Meses con 31 días / Months with 31 days
85                     return True
86             return False # ❌ Día fuera de rango / Day out of range
87     except ValueError:
88         return False # 🔞 Si no se puede convertir a número / If conversion fails, return False
89
90

```

Ilustración 65 Código Validaciones

Unidad 4

Practica 1 (Suma de Números)

Descripción: En esta actividad realizamos una Suma de 2 números mediante el uso de ventanas en la primera ventana no solicitara únicamente ingresar dos números en cajas de texto estos números serán Enviados a la segunda ventana dónde al pulsar un botón podremos realizar la suma de los mismos. El aprendizaje principal que se llevó a cabo en esta práctica fue la utilización de 2 ventanas Que están relacionadas entre sí Utilizando una técnica que nos permite asignar o crear un objeto de TKinter a una variable y controlar su comportamiento mediante esta variable.

```

1  from tkinter import * # se importa todo
2  from tkinter import messagebox
3
4
5
6  class Principal():
7      def __init__(self, master):
8          self.vetana = master # vetanatana primaria para todo el programa
9          self.vetana.title("Practica 1 Parcial 3") # 🔍 Título de la vetanatana / Sets window title
10         # self.val = validaciones1() # 🌟 Crea un objeto de la clase validaciones1 / Creates an instance of validation class
11         ancho_vetanatana = 250 # 🔍 Ancho de la vetanatana / Window width
12         alto_vetanatana = 200 # 🔍 Alto de la vetanatana / Window height
13
14         # 🌟 Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
15         # obtener dimensiones de la pantalla
16         ancho_pantalla = self.vetana.winfo_screenwidth()
17         alto_pantalla = self.vetana.winfo_screenheight()
18
19         # Calcular posición para centrar
20         x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
21         y = (alto_pantalla // 2) - (alto_vetanatana // 2)
22
23         # Aplicar geometría
24         self.vetana.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
25
26     def inicio(self):
27         # caja 1
28         Label(self.vetana, text = "Escribe un numero: ").place(x = 20, y = 20)
29         self.n1 = Entry(self.vetana)
30         self.n1.place(x = 50, y = 50)
31         # caja 2
32         Label(self.vetana, text= "Escribe un numero ").place(x = 20, y = 75)
33         self.n2 = Entry(self.vetana)
34         self.n2.place(x = 50, y = 100)
35         # botones
36         Button(self.vetana, text = "Enviar", width=15, command= self.enviar).place(x = 50, y = 130)
37         Button(self.vetana, text = "Cerrar", width=15, command= self.cerrar).place(x = 50, y = 160)
38
39         self.vetana.mainloop()
40
41     def enviar(self):
42         try:

```

```

43     caja1 = int(self.n1.get())
44     caja2 = int(self.n2.get())
45     self.n1.delete(0, END)
46     self.n2.delete(0, END)
47     # self.vetana.withdraw()
48     self.vetana.withdraw() # oculta la ventana
49     otra = Toplevel(self.vetana) # manda a llamar a la otra ventana
50     Ventana2(otra, self.vetana, caja1, caja2) # se va con la otra
51 except ValueError:
52     messagebox.showerror("Error", "Algun dato no es numero 🤔")
53     self.n1.delete(0, END)
54     self.n2.delete(0, END)
55
56 def cerrar(self):
57     self.vetana.destroy()
58
59 class Ventana2():
60     def __init__(self, master, vetana, c1, c2):
61         self.venDos = master
62         self.venDos.title("Practica 1 Parcial 3") # 📺 Título de la ventanatana / Sets window title
63         # self.val = validaciones1() # ✖ Crea un objeto de la clase validaciones1 / Creates an instance of validation class
64         ancho_vetanatana = 250 # ⚡ Ancho de la vetanatana / Window width
65         alto_vetanatana = 200 # 🔍 Alto de la vetanatana / Window height
66
67         # ✖ Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
68         # Obtener dimensiones de la pantalla
69         ancho_pantalla = self.venDos.winfo_screenwidth()
70         alto_pantalla = self.venDos.winfo_screenheight()
71
72         # Calcular posición para centrar
73         x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
74         y = (alto_pantalla // 2) - (alto_vetanatana // 2)
75
76         # Aplicar geometría
77         self.venDos.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
78         Label(self.venDos, text = "Hola mundo").place(x = 50, y = 20)
79         Button(self.venDos, text = "Regresar",width=10, command= self.regresar).place(x = 50, y = 100)
80         Button(self.venDos, text = "Sumar",width=10, command= self.sumar).place(x = 50, y = 50)
81         self.vetana = vetana
82         self.c1 = c1
83
84         self.c2 = c2
85
86     def regresar(self):
87         self.venDos.destroy()
88         self.vetana.deiconify()
89
90     def sumar(self):
91         messagebox.showinfo("Suma", f"La suma de {self.c1} y {self.c2} es: {self.c1 + self.c2}")
92
93
94
95
96
97
98 if __name__ == "__main__":
99     master = Tk()
100     app = Principal(master)
101     app.inicio()
102     master.mainloop()

```

Ilustración 66 Código Practica 1

Practica 1 Pa...

Escribe un numero:

Escribe un numero

Practica 1 Pa...

Hola mundo

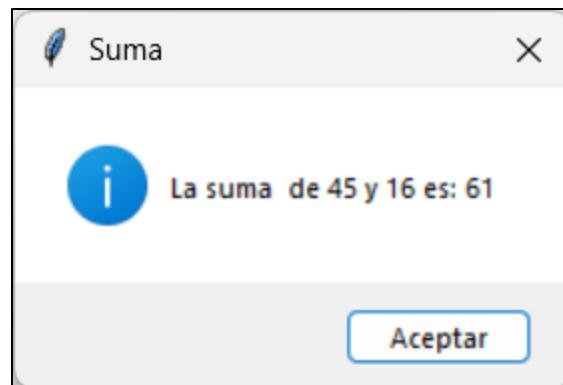


Ilustración 67 Ventanas Practica 1

Practica 2(Registro de usuarios):

Descripción: Esta práctica elaboramos la simulación de una página de logueo donde podemos registrar usuarios cada usuario según su jerarquía será las opciones que tendrá disponible para estar es la siguiente ventana, a grandes rasgos en la primera ventana se puede ir ingresar mediante el nombre de usuario y contraseña los roles que se asignaron es que el usuario administrador podrá editar modificar, agregar, y borrar usuarios de nuestra base de datos, por su parte el usuario con el rango de supervisor solamente podrá agregar usuarios, por su parte es el jefe de área solamente podrá modifica los existentes.

```

1  from tkinter import *
2  from tkinter import messagebox
3  from tkinter import ttk
4  import tkinter as tk
5  import sqlite3
6
7  def crearBaseDatos():
8      obBaseDatos = sqlite3.connect("usuarios.db")
9      cursor = obBaseDatos.cursor()
10
11     cursor.execute(''CREATE TABLE IF NOT EXISTS usuarios(
12             id INTEGER PRIMARY KEY AUTOINCREMENT,
13             usuario TEXT NOT NULL,
14             password TEXT NOT NULL
15             '')')
16
17     cursor.execute("SELECT * FROM usuarios WHERE usuario='admin'"") # busca si ya existe ese usuario "admin"
18     # si si existe regresa un TRUE, por lo tanto no agrega otro admin, ya que la condicion no se cumple
19     if not cursor.fetchone():
20         cursor.execute("INSERT INTO usuarios(usuario, password) VALUES (?,?)", ("admin","12345")) # se ingresa el dato a la base de datos
21
22     obBaseDatos.commit()
23     obBaseDatos.close() # despues de cada consulta siempre debe cerrar la base de datos
24
25 class Principal():
26     def __init__(self, master):
27         self.ventana = master # ventanata primaria para todo el programa
28         self.ventana.title("Practica 2 Parcial 3") # 📺 Título de la ventanata / Sets window title
29         # self.val = validaciones1() # ✖ Crea un objeto de la clase validaciones1 / Creates an instance of validation class
30         ancho_ventanata = 250 # ⚡ Ancho de la ventanata / Window width
31         alto_ventanata = 200 # ⚡ Alto de la ventanata / Window height
32
33         # ✖ Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
34         # obtener dimensiones de la pantalla
35         ancho_pantalla = self.ventana.winfo_screenwidth()
36         alto_pantalla = self.ventana.winfo_screenheight()
37
38         # ✖ Calcular posición para centrar (variable) ancho_ventanata: Literal[250]
39         x = (ancho_pantalla // 2) - (ancho_ventanata // 2)
40         y = (alto_pantalla // 2) - (alto_ventanata // 2)
41
42         # Aplicar geometría

```

```

43     self.vetana.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
44
45     def inicio(self):
46         # caja 1
47         Label(self.vetana, text = "Usuario ").place(x = 20, y = 20)
48         self.n1 = Entry(self.vetana)
49         self.n1.place(x = 50, y = 50)
50         # caja 2
51         Label(self.vetana, text= "Password").place(x = 20, y = 75)
52         self.n2 = Entry(self.vetana, show="x")
53         self.n2.place(x = 50, y = 100)
54         # botones
55         Button(self.vetana, text = "Validar", width=10, command= self.enviar).place(x = 30, y = 140)
56         Button(self.vetana, text = "Cerrar", width=10, command= self.cerrar).place(x = 150, y = 140)
57
58     def enviar(self):
59         u = self.n1.get()
60         p = self.n2.get()
61         # revisar en la base de datos si existe el usuario
62         con = sqlite3.connect("usuarios.db")
63         cursor = con.cursor()
64         cursor.execute("SELECT * FROM usuarios WHERE usuario=? and password=?", (u,p)) # busca si ya existe ese usuario "admin"
65         resultado = cursor.fetchone() # preguntamos si encontró el registro admin
66         con.close()
67         if resultado: # si si lo encontró pasamos a la otra ventana
68             if u == "admin" and p == "12345":
69                 self.n1.delete(0, END)
70                 self.n2.delete(0, END)
71                 # self.vetana.withdraw() # oculta la ventana 🚫
72                 otra = Toplevel(self.vetana) # manda a llamar a la otra ventana ↴
73                 Ventana2(otra, self.vetana, u) # se va con la otra
74             return 1
75
76
77     def cerrar(self):
78         self.vetana.destroy()
79
80
81
82

```

```

83
84
85 class Ventana2():
86     def __init__(self, master,vetana, u): # recibe lo que le mandaron
87         # lo que recibe guardado
88         self.venDos = master
89         self.usuario = u
90         self.vetana = vetana
91
92         self.venDos.title("Practica 1 Parcial 3") # 📺 Título de la ventanatana / Sets window title
93         # self.val = validaciones1() # 🌟 Crea un objeto de la clase validaciones1 / Creates an instance of validation class
94         ancho_vetanatana = 550 # 🖊 Ancho de la vetanatana / Window width
95         alto_vetanatana = 320 # 🖊 Alto de la vetanatana / Window height
96
97         # 🌟 Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
98         # Obtener dimensiones de la pantalla
99         ancho_pantalla = self.venDos.winfo_screenwidth()
100        alto_pantalla = self.venDos.winfo_screenheight()
101
102        # Calcular posición para centrar
103        x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
104        y = (alto_pantalla // 2) - (alto_vetanatana // 2)
105
106        # Aplicar geometría
107        self.venDos.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
108        # Label(self.venDos, text = "Hola mundo").place(x = 50, y = 20)
109        Label(self.venDos, text = "Escribe el usuario").place(x = 10, y = 10)
110        self.usuarioVenDos = Entry(self.venDos)
111        self.usuarioVenDos.place(x = 10, y = 30)
112        Label(self.venDos, text = "Escribe el Password").place(x = 150, y = 10)
113        self.contraseñaH = Entry(self.venDos)
114        self.contraseñaH.place(x = 150, y = 30)
115
116        self.us = Label(self.venDos, text = "")
117        self.us.place(x = 300, y = 10)
118        self.us.config(text = f"BIENVENIDO \n {self.usuario}")
119        # Button(self.venDos, text = "Regresar",width=10, command= self.regresar).place(x = 50, y = 100)
120        self.mostrar()
121        self.mostrar_tabla()
122        self.menus = tk.Menu(self.venDos) # objeto de la librería de tk inter
123        self.venDos.config(menu = self.menus)
124        self.archivo = tk.Menu(self.menus, tearoff = 0)

```

```

125     self.archivo.add_command(label = "Salir", command = self.salir)
126     self.archivo.add_command(label = "Modificar", command = self.modificarUsuario)
127     self.indexModificar = self.archivo.index("end")
128     self.archivo.add_command(label = "Eliminar", command = self.eliminarUsuario)
129     self.indexEliminar = self.archivo.index("end")
130     self.archivo.add_command(label = "Aregar", command = self.crearUsuario)
131     self.indexAregar = self.archivo.index("end")
132     self.menus.add_cascade(label = "Archivo", menu = self.archivo)
133     self.index = -1
134     self.archivo.entryconfig(self.indexAregar, state = "disabled")
135     self.archivo.entryconfig(self.indexModificar, state = "disabled")
136     self.archivo.entryconfig(self.indexEliminar, state = "disabled")
137     # print(self.indexAregar)
138     # print(self.indexEliminar)
139     self.roles()
140
141
142     def roles(self):
143         if self.usuario == "admin":
144             self.archivo.entryconfig(self.indexAregar, state = "normal")
145             self.archivo.entryconfig(self.indexModificar, state = "normal")
146             self.archivo.entryconfig(self.indexEliminar, state = "normal")
147
148         elif self.usuario in ("Supervisor", "supervisor"):
149             self.archivo.entryconfig(self.indexAregar, state = "normal")
150             self.archivo.entryconfig(self.indexModificar, state = "disabled")
151             self.archivo.entryconfig(self.indexEliminar, state = "disabled")
152
153         elif self.usuario in ("Jefe de area", "jefe de area"):
154             self.archivo.entryconfig(self.indexAregar, state = "disabled")
155             self.archivo.entryconfig(self.indexModificar, state = "normal")
156             self.archivo.entryconfig(self.indexEliminar, state = "disabled")
157
158     def seleccionFila(self, event):
159         try:
160             self.index = self.tabla.selection()[0] # obtengo solo la direccion que estoy seleccionando
161
162         except:
163             return
164

```

```

165     valores = self.tabla.item(self.index,"values")
166     self.usuarioVenDos.delete(0, END)
167     self.contraseñaN.delete(0, END)
168     self.usuarioVenDos.insert(0, valores[1])
169     self.contraseñaN.insert(0, valores[2])
170     # print(valores)
171     # return valores
172
173
174     def eliminarUsuario(self):
175         try:
176             self.index = self.tabla.selection()[0] # obtengo solo la direccion que estoy seleccionando
177             valores = self.tabla.item(self.index,"values")
178             # print(valores)
179             usuario = valores[1]
180             id = valores[0]
181             # print(valores[0])
182             # print("antes de interrogante")
183             if usuario == self.usuario:
184                 messagebox.showerror("Error", "No te puedes eliminar a ti mismo")
185                 return 1
186
187
188             obBaseDatos = sqlite3.connect("usuarios.db")
189             cursor = obBaseDatos.cursor()
190             cursor.execute(f"DELETE FROM usuarios WHERE id={id}")#, (id,)) # se debea agregar la coma para poder borrar dos elementos en el
191             obBaseDatos.commit()
192             obBaseDatos.close()
193
194             self.actualizarTabla()
195             self.borrarDatos("Usuario Eliminado Corretamente")
196             self.index = -1
197
198         except:
199             messagebox.showerror("Error", "Elije una fila")
200
201     def modificarUsuario(self):
202         try:
203             self.index = self.tabla.selection()[0] # obtengo solo la direccion que estoy seleccionando
204         except:

```

```

204     except:
205         messagebox.showerror("Error", "Elije un Usuario")
206         return 1
207
208     valores = self.tabla.item(self.index,"values")
209     id = valores[0]
210
211     if len(self.usuarioVenDos.get()) != 0 and len(self.contraseñaN.get()) != 0:
212         usuario = self.usuarioVenDos.get()
213         password = self.contraseñaN.get()
214         obBaseDatos = sqlite3.connect("usuarios.db")
215         cursor = obBaseDatos.cursor()
216         cursor.execute("UPDATE usuarios SET usuario=?, password=? WHERE id=?", (usuario,password, id))
217         obBaseDatos.commit()
218         obBaseDatos.close()
219         self.borrarDatos("Datos actualizados")
220         self.actualizarTabla()
221         self.index = -1
222
223
224     else:
225         messagebox.showerror("Error", "Faltan Datos")
226
227 def crearUsuario(self):
228     if len(self.usuarioVenDos.get()) != 0 and len(self.contraseñaN.get()) != 0:
229         obBaseDatos = sqlite3.connect("usuarios.db")
230         cursor = obBaseDatos.cursor()
231         cursor.execute("INSERT INTO usuarios(usuario, password) VALUES (?,?)", (self.usuarioVenDos.get(),self.contraseñaN.get()))
232         obBaseDatos.commit()
233         obBaseDatos.close()
234         self.borrarDatos("Usuario Agregado correctamente")
235
236         self.actualizarTabla()
237
238         return 0
239     else:
240         messagebox.showerror("Error", "Faltan datos")
241
242
243 def salir(self):

```

```

244     self.venDos.destroy()
245     self.vetana.destroy()
246
247
248 def actualizarTabla(self):
249
250     for i in self.tabla.get_children():
251         self.tabla.delete(i)
252
253     self.mostrar_tabla()
254
255 def mostrar_tabla(self):
256     con = sqlite3.connect("usuarios.db")
257     cursor = con.cursor()
258     cursor.execute("SELECT * FROM usuarios") # busca si ya existe ese usuario "admin"
259     resultado = cursor.fetchall() # solicita acceso a todos los registros de la base de datos
260     # print(resultado)
261     for i in resultado:
262         self.tabla.insert("", END, values = i)
263         print(i)
264
265     con.close()
266
267 def borrarDatos(self, mensaje):
268     self.usuarioVenDos.delete(0, END)
269     self.contraseñaN.delete(0, END)
270     messagebox.showinfo("Datos", mensaje)
271
272 def mostrar(self):
273     # dataGridView:
274     #   Crea la tabla (Treeview) con encabezados / Creates data table (Treeview) with headers
275     #   columnas = ("ID", "USUARIO", "PASSWORD") # Son las columnas de nuestra base de datos
276     #   self.tabla = ttk.Treeview(self.venDos, columns = columnas, show= "headings")
277     #   self.tabla.place(x = 10, y = 100, width = 350, height = 190)
278
279     #   Configura encabezados y columnas centradas / Sets up headers and centers columns
280     for col in columnas:
281         self.tabla.heading(col, text = col)
282         self.tabla.column(col, anchor="center", width = 30)

```

```

283
284     # Barras de desplazamiento vertical y horizontal / Vertical and horizontal scrollbars
285     scrollY = ttk.Scrollbar(self.venDos, orient = "vertical", command = self.tabla.yview)
286     scrollX= ttk.Scrollbar(self.venDos, orient = "horizontal", command = self.tabla.xview)
287     scrollY.place(x = 360, y = 100, height = 190)
288     scrollX.place(x = 10, y = 280, width = 350 )
289     self.tabla.bind("<<TreeviewSelect>>", self.seleccionFila) # detectar eventos en la tabla
290
291
292     # def regresar(self):
293     #     self.venDos.destroy()
294     #     self.vetana.deiconify()
295
296
297
298
299
300
301 if __name__ == "__main__":
302     crearBaceDatos() # los primero que hace es crear la base de datos
303     master = Tk()
304     app = Principal(master)
305     app.inicio()
306     master.mainloop()

```

Ilustración 68 Código Práctica 2

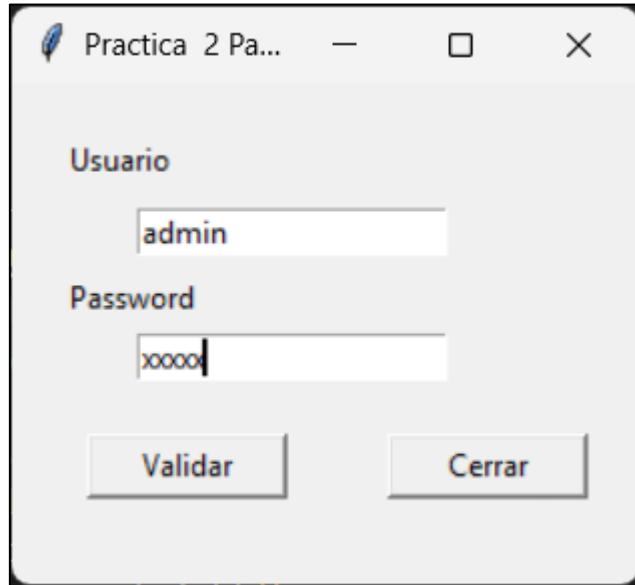


Ilustración 69 Ventana 1 Práctica 2

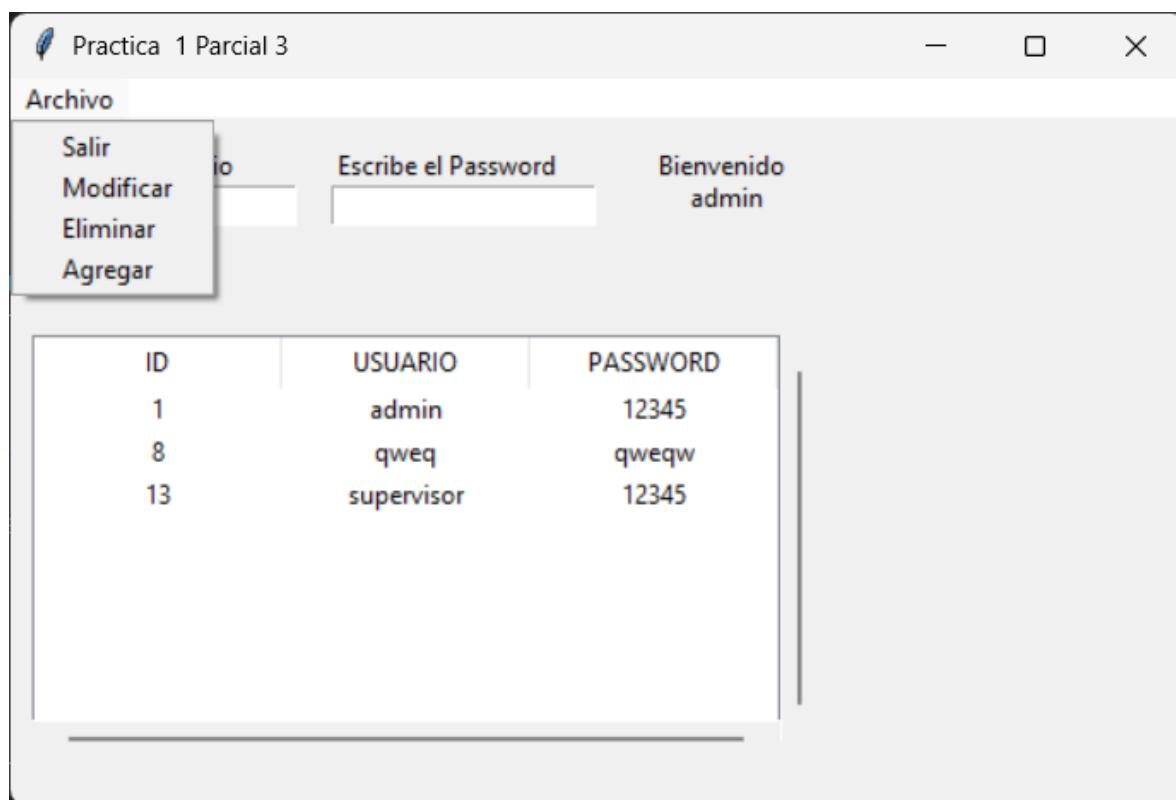


Ilustración 72 Ventana 2 Practica 2 Modo admin



Ilustración 71 Opciones de Modo Supervisor



Ilustración 70 Opciones del Modo Jefe de área

Filtrar 2 tablas...		Filas: 4			Filtrar 4 filas...	Actualizar a PRO
▼ TABLAS						
> sqlite_sequence						
▼ usuarios						
# ROWID	1	1	admin	12345		
# id	2	8	qweq	qweqw		
# usuario	3	13	supervisor	12345		
# password	4	14	Jefe de area	12345		
	+	5				

Ilustración 73 Base de datos de la Practica 2

Practica 3 (Sistema de Tienda):

Descripción: En esta práctica se creó una base de datos que cuenta con dos tablas, la primera de ellas guarda información relativa a productos mientras que la segunda conserva datos del inventario, ambas comparten un id idéntico que relaciona la información de ambas tablas, ya que estará asociado al nombre de su producto específico, para conseguir esto se optó por un diseño simple en el cual se ingresan los datos mediante el uso de cajas de texto y se visualizan utilizando una tabla, también contamos con 3 botones uno de agregar, otro para modificar y un último que pertenece a la sección de eliminar, con un mecanismo que permite deshabilitar los botones de modificar y eliminar mientras que no se haya seleccionado ningún renglón de la tabla, al igual no se aceptan registros repetidos, en caso de que haya algún registro repetido se editará el registro existente. Desde luego también se verifica que las cajas de texto no estén vacías cuando se intente agregar un nuevo producto.

```

1  from tkinter import *
2  from tkinter import messagebox
3  from tkinter import ttk
4  import tkinter as tk
5  import sqlite3
6  import random
7
8  def crearBaseDatos():
9      obBaseDatos = sqlite3.connect("tienda.db")
10     cursor = obBaseDatos.cursor()
11
12    cursor.execute('''CREATE TABLE IF NOT EXISTS productos(
13        id INTEGER PRIMARY KEY AUTOINCREMENT,
14        codigo TEXT NOT NULL,
15        producto TEXT NOT NULL,
16        precio REAL NOT NULL
17        ...)''')
18    cursor.execute('''CREATE TABLE IF NOT EXISTS almacen(
19        id INTEGER PRIMARY KEY AUTOINCREMENT,
20        codigo_producto TEXT NOT NULL,
21        stock INTEGER NOT NULL,
22        descripcion TEXT NOT NULL
23        ...)''')
24
25    # cursor.execute("SELECT * FROM productos WHERE usuario='admin'"") # busca si ya existe ese usuario "admin"
26    # # si si existe regresa un TRUE, por lo tanto no agrega otro admin, ya que la condicion no se cumple
27    # if not cursor.fetchone():
28    #     cursor.execute("INSERT INTO productos(producto, precio) VALUES (?,?)", ("admin","12345")) # se ingresa el dato a la base de datos
29
30    obBaseDatos.commit()
31    obBaseDatos.close() # despues de cada consulta siempre debe cerrar la base de datos
32
33
34
35  class Principal():
36      def __init__(self, master):
37          self.ventana = master # ventanata primaria para todo el programa
38          self.ventana.title("Practica 2 Parcial 3") # Título de la ventanata / Sets window title
39          # self.val = validaciones1() # Crea un objeto de la clase validaciones1 / Creates an instance of validation class
40          ancho_ventanata = 550 # Ancho de la ventanata / Window width
41          alto_ventanata = 500 # Alto de la ventanata / Window height

```

```

42
43     # * Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
44     # obtener dimensiones de la pantalla
45     ancho_pantalla = self.vetana.winfo_screenwidth()
46     alto_pantalla = self.vetana.winfo_screenheight()
47
48     # Calcular posición para centrar
49     x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
50     y = (alto_pantalla // 2) - (alto_vetanatana // 2)
51
52     # Aplicar geometría
53     self.vetana.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
54     self.index = -1
55
56
57     def inicio(self):
58         self.us = Label(self.vetana, text = "CRUD del Producto")
59         self.us.place(x = 5, y = 5)
60         Label(self.vetana, text = "Producto").place(x = 5, y = 30)
61         self.producto = Entry(self.vetana)
62         self.producto.place(x = 5, y = 50)
63         Label(self.vetana, text = "Descripción").place(x = 135, y = 30)
64         self.descripcion = Entry(self.vetana)
65         self.descripcion.place(x = 135, y = 50)
66         Label(self.vetana, text = "Precio").place(x = 265, y = 30)
67         self.precio = Entry(self.vetana)
68         self.precio.place(x = 265, y = 50)
69         Label(self.vetana, text = "Cantidad").place(x = 400, y = 30)
70         self.cantidad = Entry(self.vetana)
71         self.cantidad.place(x = 400, y = 50)
72         columnas = ("ID", "CÓDIGO", "PRODUCTO", "PRECIO", "DESCRIPCION", "STOCK") # Son las columnas de nuestra base de datos
73         self.tabla = ttk.Treeview(self.vetana, columns = columnas, show= "headings")
74         self.tabla.place(x = 10, y = 100, width = 480, height = 190)
75
76     # [!] Configura encabezados y columnas centradas / Sets up headers and centers columns
77     for col in columnas:
78         self.tabla.heading(col, text = col)
79         self.tabla.column(col, anchor="center", width = 30)
80

```

```

81
82     # [!] Barras de desplazamiento vertical y horizontal / Vertical and horizontal scrollbars
83     scrollry = ttk.Scrollbar(self.vetana, orient = "vertical", command = self.tabla.yview)
84     scrollx= ttk.Scrollbar(self.vetana, orient = "horizontal", command = self.tabla.xview)
85     scrollx.place(x = 480, y = 90, height = 200)
86     scrollx.place(x = 10, y = 280, width = 470 )
87     self.tabla.bind("<>TreeviewSelect>", self.seleccionFila) # detectar eventos en la tabla
88     self.btnAgregar = Button(self.vetana, text = "Agregar", command = self.agregarProducto, width = 10)
89     self.btnAgregar.place(x = 30, y = 320)
90     self.btnModificar = Button(self.vetana, text = "Modificar", command = self.modificarProducto, width = 10, state = "disabled")
91     self.btnModificar.place(x = 120, y = 320)
92     self.btnEliminar = Button(self.vetana, text = "Eliminar", command = self.eliminarProducto, width = 10, state = "disabled")
93     self.btnEliminar.place(x = 210, y = 320)
94     self.mostrarDatos()
95
96     def modificarProducto(self):
97         try:
98             self.index = self.tabla.selection()[0] # obtengo solo la dirección que estoy seleccionando
99
100            except:
101                return
102
103            valores = self.tabla.item(self.index,"values")
104            # print(valores)
105            id = valores[0]
106            producto = self.producto.get()
107            precio = self.precio.get()
108            descripcion = self.descripcion.get()
109            cantidad = self.cantidad.get()
110
111            if len(producto) != 0 and len(precio) != 0 and len(descripcion) != 0 and len(cantidad) != 0:
112
113                codigo = producto[:2].upper() + str(random.randint(0,100)) + descripcion[0].upper()
114                obBaseDatos = sqlite3.connect("tienda.db")
115                cursor = obBaseDatos.cursor()
116                cursor.execute("UPDATE productos SET codigo=?, producto=?, precio=? WHERE id=?", (codigo,producto,precio, id))
117                cursor.execute("UPDATE almacen SET codigo_producto=?, stock=?, descripcion=? WHERE id=?", (codigo,cantidad,descripcion, id))
118                obBaseDatos.commit()
119                obBaseDatos.close()
120                self.actualizarTabla()

```

```

120         self.borrarCajas()
121         self.btnAgregar.config(state= "normal")
122         self.btnEliminar.config(state="disabled")
123         self.btnModificar.config(state="disabled")
124     else:
125         messagebox.showerror("Error", "Faltan datos")
126
127
128
129
130     def eliminarProducto(self):
131         try:
132             self.index = self.tabla.selection()[0] # obtengo solo la direccion que estoy seleccionando
133
134         except:
135             return
136
137         valores = self.tabla.item(self.index,"values")
138         print(valores)
139         id = valores[0]
140         obBaseDatos = sqlite3.connect("tienda.db")
141         cursor = obBaseDatos.cursor()
142         cursor.execute("DELETE FROM productos WHERE id=?",(id,))
143         cursor.execute("DELETE FROM almacen WHERE id=?",(id,))
144         self.actualizarTabla()
145         self.borrarCajas()
146         self.btnAgregar.config(state= "normal")
147         self.btnEliminar.config(state="disabled")
148         self.btnModificar.config(state="disabled")
149         obBaseDatos.commit()
150         obBaseDatos.close()
151
152
153
154     def seleccionFila(self, event):
155         self.borrarCajas()
156         try:
157             self.index = self.tabla.selection()[0] # obtengo solo la direccion que estoy seleccionando
158
159         except:
160             return
161
162         valores = self.tabla.item(self.index,"values")
163         self.producto.insert(0,valores[2])
164         self.precio.insert(0, valores[3] )
165         self.descripcion.insert(0,valores[4])
166         self.cantidad.insert(0, valores[5])
167         self.btnAgregar.config(state= "disabled")
168         self.btnEliminar.config(state="normal")
169         self.btnModificar.config(state="normal")
170
171     def mostrarBdatos(self):
172         obBaseDatos = sqlite3.connect("tienda.db")
173         cursor = obBaseDatos.cursor()
174         cursor.execute('''
175             SELECT productos.id,
176                 productos.codigo,
177                 productos.producto,
178                 productos.precio,
179                 almacen.descripcion,
180                 almacen.stock
181             FROM productos
182             INNER JOIN almacen
183             ON productos.codigo = almacen.codigo_producto
184             ...
185
186         ''')
187         datos = cursor.fetchall()
188         # print(datos)
189         for i in datos:
190             self.tabla.insert("",END, values = i)
191         obBaseDatos.commit()
192         obBaseDatos.close()
193
194     def verificar(self):
195         precioTotal = 0.0
196         stockInt = 0
197         producto = self.producto.get()

```

```
198     descripcion = self.descripcion.get()
199     precio = self.precio.get()
200     stock = self.cantidad.get()
201     if len(precio) != 0 and len(descripcion) != 0 and len(producto) != 0 and len(stock) != 0:
202         obBaseDatos = sqlite3.connect("tienda.db")
203         cursor = obBaseDatos.cursor()
204         cursor.execute('''
205             SELECT *
206             FROM productos
207             WHERE producto=?''',(producto,))
208         resultado = cursor.fetchone()
209         obBaseDatos.commit()
210         obBaseDatos.close()
211         print(resultado)
212
213     if resultado:
214         return resultado[0]
215     return False
216
217
218
219 def agregarProducto(self):
220     if self.verificar():
221         # messagebox.showinfo("Ya", "Este producto ya existe")
222         id = self.verificar()
223         print(id)
224         precioFloat = 0.0
225         stockInt = 0
226         producto = self.producto.get()
227         descripcion = self.descripcion.get()
228         precio = self.precio.get()
229         stock = self.cantidad.get()
230         obBaseDatos = sqlite3.connect("tienda.db")
231         cursor = obBaseDatos.cursor()
232         cursor.execute("UPDATE productos SET precio=? WHERE id=?", (precio, id))
233         cursor.execute("UPDATE almacen SET stock=?, descripcion=? WHERE id=?", (stock,descripcion, id))
234         obBaseDatos.commit()
235         obBaseDatos.close()
236         self.actualizarTabla()
237
238
239     self.borrarCajas()
240
241 else:
242     precioFloat = 0.0
243     stockInt = 0
244     producto = self.producto.get()
245     descripcion = self.descripcion.get()
246     precio = self.precio.get()
247     stock = self.cantidad.get()
248     if len(precio) != 0 and len(descripcion) != 0 and len(producto) != 0 and len(stock) != 0:
249         precioFloat = float(precio)
250         stockInt = int(stock)
251         codigo = producto[:].upper() + str(random.randint(0,100)) + descripcion[0].upper()
252         obBaseDatos = sqlite3.connect("tienda.db")
253         cursor = obBaseDatos.cursor()
254         cursor.execute("INSERT INTO productos(codigo,producto,precio) VALUES (?,?,?)", (codigo,producto,precioFloat))
255         cursor.execute("INSERT INTO almacen(codigo_producto,descripcion,stock) VALUES (?,?,?)", (codigo,descripcion, stockInt))
256         obBaseDatos.commit()
257         obBaseDatos.close()
258         self.actualizarTabla()
259         self.borrarCajas()
260     else:
261         messagebox.showerror("Error","Faltan Datos")
262
263
264 def actualizarTabla(self):
265     for i in self.tabla.get_children():
266         self.tabla.delete(i)
267
268     self.mostrarDatos()
269
270 def borrarCajas(self):
271     self.precio.delete(0,END)
272     self.producto.delete(0,END)
273     self.descripcion.delete(0,END)
274     self.cantidad.delete(0,END)
275     return 0
276
277
278 if __name__=="__main__":
279     crearBasesDatos() # los primero que hace es crear la base de datos
280     master = Tk()
281     app = Principal(master)
282     app.inicio()
283     master.mainloop()
```

Ilustración 74 Código Práctica 3

Practica 2 Parcial 3

CRUD del Producto

Producto	Descripcion	Precio	Cantidad

ID	CODIGO	PRODUCTO	PRECIO	DESCRIPCION	STOCK
8	AZ91M	azucar	56.0	morena	6
9	SA33M	sal	45.0	mar	5

Agregar **Modificar** **Eliminar**

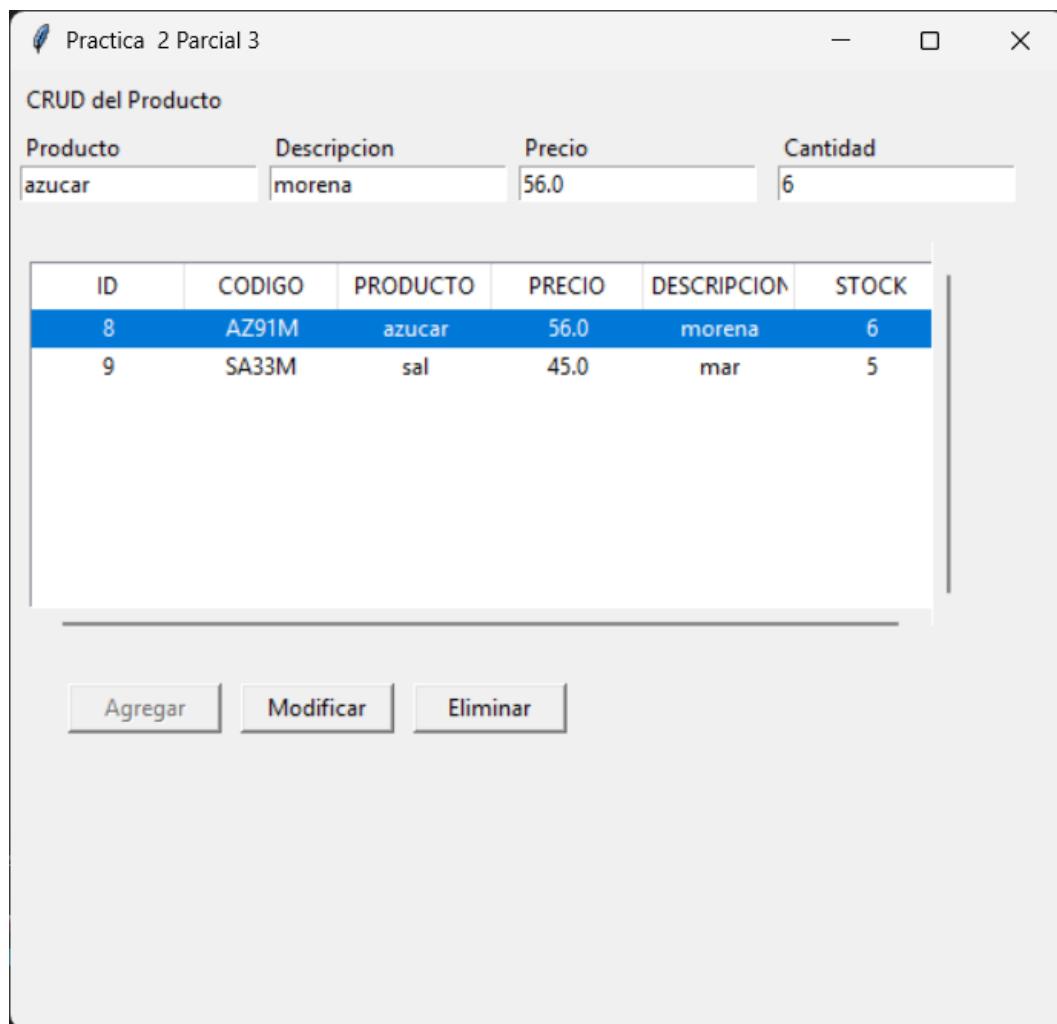


Ilustración 75 Ventana Practica 3

tienda.db		Filas: 2					Filtrar 2 filas...	Actualizar a PRO
TABLAS			id	codigo_p...	stock	descripc...		
>	almacen		1	8	AZ91M	6	morena	
>	productos		2	9	SA33M	5	mar	
	sqlite_sequence		+	3				

tienda.db		Filas: 2					Filtrar 2 filas...	Actualizar a PRO
TABLAS			id	codigo	producto	precio		
>	almacen		1	8	AZ91M	azucar	56	
>	productos		2	9	SA33M	sal	45	
	sqlite_sequence		+	3				

Ilustración 76 Tablas de la base de datos Practica 3

Tarea 1 (Mostrar numero en listBox):

Descripción: En esta tarea se hace uso de dos ventanas, la primera de ellas nos solicitará ingresar un número el cual se enviará a la segunda ventana que no mostrará ese número repetida la cantidad de veces que el número que representa en una listBox, para entender un poco mejor el mecanismo un ejemplo sería si nosotros escribimos en la primera ventana el número 10 entonces en la listBox nos aparecerá escrito 10 veces el número 10.

```

1  ...
2  hacer un programa en tkinter que en una ventana mediante
3  una caja de texto lea un numero, ese numero se enviara a
4  otra ventana donde en un listBox, mostrara ese numero, el numero de veces
5  ...
6
7  from tkinter import *
8  from tkinter import messagebox
9  from tkinter import ttk
10
11 class Principal():
12     def __init__(self, master):
13         self.vetana = master # vetanatana primaria para todo el programa
14         self.vetana.title("Practica 1 Parcial 3") # ➔ Título de la vetanatana / Sets window title
15         # self.val = validaciones1() # ✖ Crea un objeto de la clase validaciones1 / Creates an instance of validation class
16         ancho_vetanatana = 250 # ↘ Ancho de la vetanatana / Window width
17         alto_vetanatana = 200 # ↙ Alto de la vetanatana / Window height
18
19         # ✖ Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
20         # Obtener dimensiones de la pantalla
21         ancho_pantalla = self.vetana.winfo_screenwidth()
22         alto_pantalla = self.vetana.winfo_screenheight()
23
24         # Calcular posición para centrar
25         x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
26         y = (alto_pantalla // 2) - (alto_vetanatana // 2)
27
28         # Aplicar geometría
29         self.vetana.geometry(f"{{ancho_vetanatana}}x{{alto_vetanatana}}+{x}+{y}")
30
31     def inicio(self):
32         Label(self.vetana, text = "Número: ").place(x = 100, y = 50)
33         self.nume1 = Entry(self.vetana)
34         self.nume1.place(x = 60, y = 100)
35         Button(self.vetana, text = "Enviar", command = self.enviar).place(x = 100, y = 150)
36         Button(self.vetana, text = "Salir", command = self.destruir).place(x = 200, y = 150)
37
38     def enviar(self):
39         try:
40             if len(self.nume1.get()) <= 0:
41                 messagebox.showerror("Error", "Campo Básico")
42             return 1

```

```

43
44     numero = int(self.num1.get())
45     self.num1.delete(0, END)
46     self.vetana.withdraw() # oculto esta ventana
47     nuevaVen = Toplevel(self.vetana)
48     Ventana_lista(nuevaVen, self.vetana, numero)
49
50 except ValueError:
51     messagebox.showerror("Error", "No son numeros")
52     self.
53
54 def destruir(self):
55
56     self.vetana.destroy()
57
58 class Ventana_lista():
59     def __init__(self, master, vetana, dato):
60         self.ventaPrimaria = vetana
61         self.numero = dato
62         self.vetanaSec = master
63         self.vetanaSec.title("Resultado de numero") # ☈ Título de la ventana / Sets window title
64         # self.val = validaciones1() # ✖ Crea un objeto de la clase validaciones1 / Creates an instance of validation class
65         ancho_vetanatana = 250 # ↗ Ancho de la ventana / Window width
66         alto_vetanatana = 200 # ↗ Alto de la ventana / Window height
67
68         # ✖ Obtiene el ancho y alto de la pantalla en milímetros / Gets screen width and height in millimeters
69         # Obtener dimensiones de la pantalla
70         ancho_pantalla = self.vetanaSec.winfo_screenwidth()
71         alto_pantalla = self.vetanaSec.winfo_screenheight()
72
73         # Calcular posición para centrar
74         x = (ancho_pantalla // 2) - (ancho_vetanatana // 2)
75         y = (alto_pantalla // 2) - (alto_vetanatana // 2)
76
77         # Aplicar geometría
78         self.vetanaSec.geometry(f"{ancho_vetanatana}x{alto_vetanatana}+{x}+{y}")
79         self.inicio()
80
81     def inicio(self):
82         Label(self.vetanaSec, text = "Resultados de la Operacion").place(x = 50, y = 10)
83
83         self.milista = Listbox(self.vetanaSec, height = 10, width = 8, bg = "white")
84         self.milista.place(x = 80, y = 30)
85         Button(self.vetanaSec, text = "Regresar", command = self.volver).place(x = 10, y = 50)
86         self.ingresarDatos()
87
88     def ingresarDatos(self):
89
90         for i in range(self.numero):
91             self.milista.insert(END, self.numero)
92
93     def volver(self):
94         self.vetanaSec.destroy()
95         self.ventaPrimaria.deiconify()
96
97
98 if __name__ == "__main__":
99     master = Tk()
100    app = Principal(master)
101    app.inicio()
102    master.mainloop()

```

Ilustración 77 Código Tarea 1

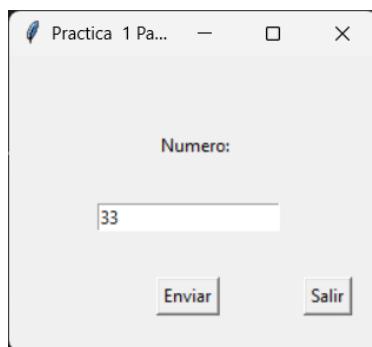


Ilustración 78 Ventana 1 Tarea 1

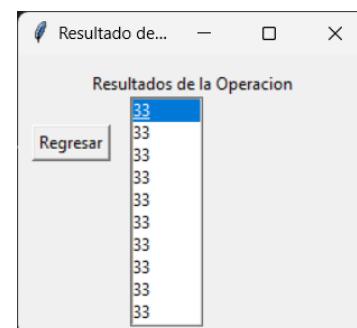


Ilustración 79 Ventana 2 Tarea 1