



PRÁCTICO 1

PUNTEROS LETALES

Jorge Luis Esteves Salas

Fernando Navia

Joel Dalton Montero

Ana Laura Cuellar

Weimar Valda

Leonel Eguez Camargo

Alejandro Hurtado Rodas

Dirigido por el docente:

JIMMY REQUENA LLORENTTY

Materia:

Programación II

MIS PRIMERAS PRUEBAS DE REPLIT

Después de seguir los pasos del texto guía en la plataforma pude configurar mi Replit con mi repositorio tuve unas complicaciones y todavía no puedo configurar el token de acceso correctamente pero la sincronización de archivos entre funciona bien:

```
~/workspace: git pull

~/workspace/pruebasProg2$ git commit -m "prueba de subida 2"
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
(use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ../.replit
        deleted:    ../PruebasProg2/clese02_prueba.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        clese02_prueba.py

no changes added to commit (use "git add" and/or "git commit -a")
~/workspace/pruebasProg2$ git push origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (15/15), 1.76 KiB | 1.76 MiB/s, done.
Total 15 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Alejandr026/prog2-aalejandrohurtado
   2dab68a..d6c3f11  main -> main
~/workspace/pruebasProg2$ ls
clese02_prueba.py  pruebasubida.md
~/workspace/pruebasProg2$ cd..
bash: cd.: command not found
~/workspace/pruebasProg2$ cd.
```

UNIVERSIDAD PRIVADA
DOMINGO SAVIO

Añadi y elimine archivos tanto en el repositorio como en el replit y funciona bien :

```
~/workspace: git pull
~/workspace$ git add linkUtiles.md
~/workspace$ git commit -m "update del archivo"
[main c161965] update del archivo
1 file changed, 5 insertions(+)
create mode 100644 linkUtiles.md
~/workspace$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Alejandro026/prog2-alejandrohurtado
d6c3f11..c161965 main -> main
~/workspace$ git pull
Already up to date.
~/workspace$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (2/2), 874 bytes | 874.00 KiB/s, done.
From https://github.com/Alejandro026/prog2-alejandrohurtado
c161965..39836c0 main -> origin/main
Updating c161965..39836c0
Fast-forward
 Links Utiles | 3 ---
1 file changed, 3 deletions(-)
delete mode 100644 Links Utiles
~/workspace$
```

APRENDIENDO A LLAMAR FUNCIONES

La clase de centro en aprender a usar las funciones y llamarlas para poder optimizar el código use de base los ejercicios que nos dio de tarea para para modificarlos y crear funciones que posteriormente llame


Versión original de la tarea

```
Shell | edad.py | funcionesPrueba.py | funciones.py | adivinaNumero.py x | pruebaNumSecreto.py | + | :
ejerciciosProrg2 > acti1 > adivinaNumero.py > ...
1 #Adivina el numero secreto
2
3 num = 7
4
5 print("Adivina el numero secreto del 1 al 10\n")
6 num_user = int(input("\nIngresa el numero: "))
7 if num_user == num:
8     print("\nFelicidades adivinaste el numero secreto")
9 else:
10    print("\nLo siento no adivinaste el numero secreto\n")
11
12 print ("____Alejandro Hurtado____")
13
14
```

Función creada a partir el código original

```
17
18 #Adivina el numero secreto
19
20 def num_secreto(num):
21     print("Adivina el numero secreto del 1 al 10\n")
22     num_user = int(input("Ingrese el numero: "))
23     if num_user == num:
24         return "\nFelicitades, adivinaste el numero secreto"
25     else:
26         return "\nLo siento, no adivinaste el numero secreto\n"
27
28 print ("____Alejandro Hurtado____")
```

Prueba llamando a la función desde otro archivo de código de Python



```
Shell  edad.py  funcionesPrueba.py  funciones.py  adivinaNumero.py  pruebaNumSecreto.py x +
ejercicosProrg2 > clase-08-07-2025 > pruebaNumSecreto.py > ...
1 from funciones import num_secreto
2
3 print ("Ingrese un numero secreto del 1 al 10\n")
4 num = int(input("Ingrese el numero: "))
5 print (num_secreto(num))
```

Demostración del Funcionamiento en la terminal de Replit

```
Shell x edad.py funcionesPrueba.py funciones.py adivinaNumero.py pruebaNumSecreto.py +
~/workspace/ejerciciosProrg2/clase-08-07-2025: python pruebaNumSecreto.py

~/workspace$ cd ejerciciosProrg2/clase-08-07-2025/
~/.../ejerciciosProrg2/clase-08-07-2025$ python pruebaNumSecreto.py
____Alejandro Hurtado____
Ingrese un numero secreto del 1 al 10

Ingrese el numero: 3
Adivina el numero secreto del 1 al 10

Ingrese el numero: 7

Lo siento, no adivinaste el numero secreto

~/.../ejerciciosProrg2/clase-08-07-2025$
```

PRUEBAS DE LISTAS O VECTORES

Alejandro Hurtado Rodas – 10/07/2025 – 17:34

Aprendí a utilizar y guardar datos en una lista para que sean más fáciles de manejar y cambiar utilizando las listas en python, para el primer ejercicio cree un código simple en el que una lista contiene mis tres comidas favoritas y luego las imprime para posteriormente solicitar al usuario que cambie la primera comida de la lista y la vuelva a imprimir en la terminal.

```
listaNombreEstu.py listaNotas.py listaComidaFav.py x
ejerciciosProrg2 > acti2 > listaComidaFav.py > ...
1  #Lista donde se guardan las comidas
2  comida_fav = ["Salteña", "Cuñape", "Sopa de Mani"]
3  print(comida_fav)
4
5  print("\nMi segunda comida favorita es " + comida_fav[1])
6  print("\nCambiar tu primera comida favorita")
7  nueva_comida = input("\nIngresa tu nueva comida favorita:")
8  comida_fav[0] = nueva_comida
9  print(comida_fav)
10 cantidad_comida = len(comida_fav)
11 print(f"\nLa cantidad de comidas favoritas que tienes es de {cantidad_comida}")
```

```

PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/Python39-32/python.exe c:\clone\prog2-alejandrohurtado/ejerciciosProrg2/acti2/listaComidaFav.py
['Salteña', 'Cuñape', 'Sopa de Mani']

Mi segunda comida favorita es Cuñape

Cambiar tu primera comida favorita

Ingresa tu nueva comida favorita:milanesa
['milanesa', 'Cuñape', 'Sopa de Mani']

La cantidad de comidas favoritas que tienes es de 3
PS C:\clone\prog2-alejandrohurtado>

```

El segundo ejercicio consiste en tener una lista de nombres e imprimir los nombres con un mensaje de bienvenida usando un bucle For

```

listaNombreEstu.py X listaNotas.py listaComidaFav.py
ejerciciosProrg2 > acti2 > listaNombreEstu.py > ...
1 lista_nom = ["nom1", "nom2", "nom3", "nom4", "nom5"]
2
3 for i in lista_nom:
4     print(f"Bienvenido al equipo {i} ")
5
PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/Python39-32/python.exe c:\clone\prog2-alejandrohurtado/ejerciciosProrg2/acti2/listaNombreEstu.py
Bienvenido al equipo nom1
Bienvenido al equipo nom2
Bienvenido al equipo nom3
Bienvenido al equipo nom4
Bienvenido al equipo nom5
PS C:\clone\prog2-alejandrohurtado>

```

El tercer ejercicio consiste en tener una lista de notas de estudiantes, en mi caso puse 10 en todo para que sea muy fácil saber los resultados, y con esas notas sumar para conseguir el total y sacar la media de las notas pero sin usar las funciones de suma de python en su lugar se debe usar un bucle for e imprimir los resultados

```
ejercicosProrg2 > acti2 > listaNotas.py > ...
```

```
1  mis_notas = [10, 10, 10, 10, 10]
2  sum_total = 0
3  for i in mis_notas:
4      sum_total = sum_total + i
5
6  promedio = sum_total / len(mis_notas)
7  print(f"La suma total de todas las notas es de: {sum_total}")
8  print(f"El promedio de las notas es de: {promedio}")
```

```
PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/Python313
clone/prog2-alejandrohurtado/ejercicosProrg2/acti2/listaNotas.py
La suma total de todas las notas es de: 50
El promedio de las notas es de: 10.0
PS C:\clone\prog2-alejandrohurtado> 
```

USANDO FUNCIONES Y LISTAS INVERTIDAS

Alejandro Hurtado – 10/07/2025 – 20:43

Los ejercicios de la clase de hoy se trataron de hacer diversas cosas con las listas una de ellas fue tomar una lista existente y darle la vuelta o invertir su contenido para posteriormente imprimirlo usando unas simples lineas de código de una manera moderna evitando el uso de modos antiguos más extensos:

```
ejercicosProrg2 > clase3 > listaInvertida.py > ...
```

```
1  #Crear una lista invertia
2
3  def invertir_lista (list_var):
4      return list_var[::-1]
5
6  lista = [1, 2, 3]
7
8  print(invertir_lista(lista))
9
10 
```

```
● PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/P
clone/prog2-alejandrohurtado/ejercicosProrg2/clase3/listaInvertida.py
[1, 2, 3]
● [3, 2, 1]
● PS C:\clone\prog2-alejandrohurtado> 
```

También realizamos un contador de caracteres para una palabra (hola mundo) en este caso usando un bucle for y un contador simple:

```
ejerciciosProng2 > clase3 > contadorElementos.py > ...
1  #Contador de cuntas veces aparece un elementon de una lista
2
3  palabra = "hola mundo"
4  contador = 0
5
6  for i in palabra:
7      if i == "o":
8          contador = contador + 1
9  print(f"El numero de veces que se repite la o es de: {contador}")

PS C:\clone\prog2-alejandrohurtado> & C:/Users/HP/AppData/Local/Programs/Python/Python311/python.exe C:\clone\prog2-alejandrohurtado\ejerciciosProng2\clase3\contadorElementos.py
El numero de veces que se repite la o es de: 2
PS C:\clone\prog2-alejandrohurtado> 
```

PROBANDO DIFERENTES FUNCIONES DE PYTHON

Alejandro Hurtado Rodas – 10/07/2025 – 21:35

Continuando con la clase probamos diferentes funciones así como vimos que son las funciones recursivas y como usarlas con los factoriales para aprender su funcionamiento:

```
Ejercicios Docente > sumaNotas > appendVsInsert.py > ...
1  # Creamos una lista vacía
2  numeros = []
3
4  # append agrega al final
5  numeros.append(10)
6  numeros.append(20)
7  print("Usando append:", numeros) # [10, 20]
8
9  # insert agrega en una posición específica
10 numeros.insert(1, 15) # insertar en índice 1 el valor 15
11 print("Usando insert:", numeros) # [10, 15, 20]
12
13
14 # Queremos duplicar cada número
15 numeros = [1, 2, 3, 4]
16 duplicados = list(map(lambda x: x * 5.5, numeros))
17 print("Usando map:", duplicados) # [2, 4, 6, 8]
18
```


Ejercicios Docente > sumaNotas > contarElemento.py > ...

```
1  # Definición de la función
2  def contar_elemento(lista, elemento_buscado):
3      contador = 0
4      for elemento in lista:
5          if elemento == elemento_buscado:
6              contador += 1
7      return contador
8
9  # -----
10 # 🖍 Casos de prueba con assert
11 # -----
12
13 print("\nProbando contar_elemento...")
14
15 assert contar_elemento([1, 2, 3, 2, 4, 2], 2) == 3
16 assert contar_elemento(["a", "b", "a", "c", "a"], "a") == 3
17 assert contar_elemento(["sol", "luna", "estrella"], "martes") == 0
18 assert contar_elemento([], 5) == 0
19 assert contar_elemento([True, False, True, True], True) == 3
20
21 print("¡Pruebas para contar_elemento pasaron! ✅")
22
```

Ejercicios Docente > sumaNotas > contarElementoConCount.py > contar_elemento

```
1  def contar_elemento(lista, elemento_buscado):
2      return lista.count(elemento_buscado)
3  print("\nProbando contar_elemento con .count()...")
4
5  assert contar_elemento([1, 2, 3, 2, 4, 2], 2) == 3
6  assert contar_elemento([1, 2, 3, 2, 4, 2], 1) == 1
7  assert contar_elemento(["a", "b", "a", "c", "a"], "a") == 3
8  assert contar_elemento(["sol", "luna", "estrella"], "martes") == 0
9  assert contar_elemento([], 5) == 0
10 assert contar_elemento([True, False, True, True], True) == 3
11
12 print("¡Pruebas con .count() pasaron! ✅")
13
```

Ejercicios Docente > sumaNotas > encontrarMayor.py > ...

```
1  # Definición de la función
2  def encontrar_mayor(lista_numeros):
3      # Caso especial: lista vacía
4      if not lista_numeros:
5          return None
6
7      # Paso 1: El primer "campeón"
8      mayor_temporal = lista_numeros[0]
9
10     # Paso 2-4: Recorrer la lista para buscar al más grande
11     for elemento in lista_numeros:
12         if elemento > mayor_temporal:
13             mayor_temporal = elemento
14
15     # Paso 5: Devolver el campeón
16     return mayor_temporal
17
18     # -----
19     # 🟢 Casos de prueba con assert
20     # -----
21
22     print("Probando encontrar_mayor...")
23
24     assert encontrar_mayor([1, 5, 3, 9, 2]) == 9
25     assert encontrar_mayor([-10, -5, -3, -20]) == -3
26     assert encontrar_mayor([7, 7, 7, 7]) == 7
27     assert encontrar_mayor([]) == None # lista vacía
28     assert encontrar_mayor([42]) == 42 # un solo elemento
29
30     print("¡Pruebas para encontrar_mayor pasaron! ✅")
31
```

Ejercicios Docente > sumaNotas > factorial.py > ...

```
1  # Definición de la función recursiva
2  def factorial(n):
3      if n < 0:
4          raise ValueError("❌ El factorial no está definido para números negativos.")
5      elif n == 0 or n == 1:
6          return 1
7      else:
8          return n * factorial(n - 1)
9
10     assert factorial(0) == 1
11     assert factorial(1) == 1
12     assert factorial(5) == 120
13     assert factorial(6) == 720
14     print("¡Pruebas de factoriales pasaron! ✅")
15
16     # Solicita un número al usuario
17     try:
18         numero = int(input("Ingrese un número entero para calcular su factorial: "))
19         resultado = factorial(numero)
20         print(f"El factorial de {numero} es: {resultado}")
21     except ValueError as e:
22         print(e)
23
```

Ejercicios Docente > sumaNotas > factorialFor.py > ...

```
1  # Definición de la función usando un for
2  def factorial(n):
3      if n < 0:
4          raise ValueError("❌ El factorial no está definido para números negativos.")
5
6      resultado = 1
7      for i in range(2, n + 1):
8          resultado *= i
9      return resultado
10
11
12 # Solicita un número al usuario
13 try:
14     numero = int(input("Ingrese un número entero para calcular su factorial: "))
15     resultado = factorial(numero)
16     print(f"El factorial de {numero} es: {resultado}")
17 except ValueError as e:
18     print(e)
```

Ejercicios Docente > sumaNotas > factorialFuncToolsReduce.py > ...


```
1  from functools import reduce
2
3  def factorial(n):
4      if n < 0:
5          raise ValueError("❌ El factorial no está definido para números negativos.")
6      return 1 if n == 0 else reduce(lambda x, y: x * y, range(1, n + 1))
7
8  assert factorial(0) == 1
9  assert factorial(1) == 1
10 assert factorial(5) == 120
11 assert factorial(6) == 720
12 print("¡Pruebas de factoriales pasaron! ✅")
13
14 # Solicita un número al usuario
15 try:
16     numero = int(input("Ingrese un número entero para calcular su factorial: "))
17     resultado = factorial(numero)
18     print(f"El factorial de {numero} es: {resultado}")
19 except ValueError as e:
20     print(e)
```


Ejercicios Docente > sumaNotas > factorialMath.py > ...

```
1 import math
2
3 def factorial(n):
4     if n < 0:
5         raise ValueError("❌ El factorial no está definido para números negativos.")
6     return 1 if n == 0 else math.prod(range(1, n + 1))
7
8 def mostrar_factorial(n):
9     if n < 0:
10        print("❌ El factorial no está definido para números negativos.")
11        return
12
13        pasos = ' x '.join(str(i) for i in range(1, n + 1)) if n > 0 else "1"
14        resultado = factorial(n)
15        print(f"{pasos} = {resultado}")
16
17 # -----
18 # Ejecución interactiva
19 # -----
20 try:
21     numero = int(input("Ingrese un número entero para calcular su factorial: "))
22     mostrar_factorial(numero)
23 except ValueError:
24     print("❌ Entrada inválida. Por favor ingresa un número entero.")
```

Ejercicios Docente > sumaNotas > forTradicionalVsPythonico.py > ...

```
1 #CALCULO DEL CUADRADO DE LOS ELEMENTOS DE UNA LISTA DEL 0 AL 4
2 cuadrados = []
3 # TRADICIONAL
4 for x in range(5):
5     cuadrados.append(x * x)
6 print(cuadrados) # [0, 1, 4, 9, 16]
7 #PYTHONICO
8 cuadrados = [x * x for x in range(5)]
9 print(cuadrados) # [0, 1, 4, 9, 16]
10
```

Ejercicios Docente > sumaNotas >  invertirLista.py > ...

```
1  # Definición de la función
2  def invertir_lista(lista_original):
3      lista_invertida = []
4
5      # Recorremos la lista desde el final hasta el inicio
6      for i in range(len(lista_original) - 1, -1, -1):
7          lista_invertida.append(lista_original[i])
8
9      return lista_invertida
10
11 # -----
12 #  Casos de prueba con assert
13 # -----
14
15 print("\nProbando invertir_lista...")
16
17 lista_prueba = [1, 2, 3, 4, 5]
18 lista_resultante = invertir_lista(lista_prueba)
19
20 assert lista_resultante == [5, 4, 3, 2, 1], "❌ Error en inversión"
21 assert lista_prueba == [1, 2, 3, 4, 5], "❌ ¡La lista original fue modificada!"
22 assert invertir_lista(["a", "b", "c"]) == ["c", "b", "a"]
23 assert invertir_lista([]) == []
24
25 print("¡Pruebas para invertir_lista pasaron! ✅")
26
```

Ejercicios Docente > sumaNotas >  invertirListaSlicing.py >  invertir_lista

```
1  def invertir_lista(lista_original):
2      # lista[inicio:fin:paso]
3      return lista_original[::-1]
4      # Esto usa slicing con paso negativo
5      # inicio: índice desde donde empezar (incluye este elemento).
6      # fin: índice hasta donde cortar (no incluye este elemento).
7      # paso: cuántos pasos avanzar (puede ser negativo).
8
9
10 # Pruebas
11 print("\nProbando invertir_lista con slicing...")
12
13 lista_prueba = [1, 2, 3, 4, 5]
14 lista_resultante = invertir_lista(lista_prueba)
15
16 assert lista_resultante == [5, 4, 3, 2, 1], "❌ Error en inversión con slicing"
17 assert lista_prueba == [1, 2, 3, 4, 5], "❌ ¡La lista original fue modificada!"
18 assert invertir_lista(["a", "b", "c"]) == ["c", "b", "a"]
19 assert invertir_lista([]) == []
20
21 print("¡Pruebas con slicing pasaron! ✅")
```

Ejercicios Docente > sumaNotas > sumaNotas.py > ...

```
1  # Crear una lista con notas numéricas
2  mis_notas = [85.5, 90, 78, 88.5, 95, 82]
3
4  # Inicializar variable para la suma
5  suma_total = 0
6
7  # Usar bucle for para calcular la suma total sin usar sum()
8  for nota in mis_notas:
9      suma_total += nota
10
11 # Calcular el promedio
12 promedio = suma_total / len(mis_notas)
13
14 # Imprimir resultados de forma clara
15 print(f"Suma total de las notas: {suma_total}")
16 print(f"Promedio de las notas: {promedio:.2f}")
17
18 # -----
19 # Validación con pruebas assert
20 # -----
21
22 # Cálculo esperado usando sum() como referencia
23 suma_esperada = sum(mis_notas)
24 promedio_esperado = suma_esperada / len(mis_notas)
25
26 # Pruebas
27 assert suma_total == suma_esperada, f"❌ Error: la suma debería ser {suma_esperada}"
28 assert promedio == promedio_esperado, f"❌ Error: el promedio debería ser {promedio_esperado:.2f}"
29
30 print("OK ¡Todo correcto! Las validaciones pasaron exitosamente.")
31
```

Ejercicios Docente > sumaNotas > sumaPythonica.py > ...

```
1  # Definición de la función para sumar elementos (versión pythonica)
2  def sumar_elementos(lista_numeros):
3      # return sum(lista_numeros)
4      return sum([x for x in lista_numeros])
5
6  # Casos de prueba con assert
7  print("Probando sumar_elementos...")
8
9  assert sumar_elementos([1, 2, 3, 4, 5]) == 15
10 assert sumar_elementos([10, -2, 5]) == 13
11 assert sumar_elementos([]) == 0 # ¡Importante probar con una lista vacía!
12 assert sumar_elementos([100]) == 100
13
14 print("¡Pruebas para sumar_elementos pasaron! ✅")
15
```