**ESCUELA POLITÉCNICA NACIONAL**

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

# ALGORITMOS Y ESTRUCTURAS DE DATOS (TSDS)



ASIGNATURA: ALGORITMOS Y ESTRUCTURAS DE DATOS

PROFESOR: Ing. Lorena Chulde

PERÍODO ACADÉMICO: 2024-B

# TAREA Grupal

|  |
| --- |
|  |

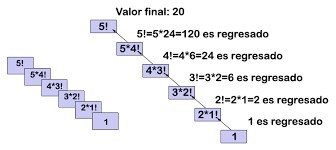
**TÍTULO:**

**FUNCIONES RECURSIVAS**

**Nombres de los estudiantes:**

Tapia Alan y Zapata Felipe

|  |
| --- |
|  |



|  |
| --- |
|  |

**2024-B**

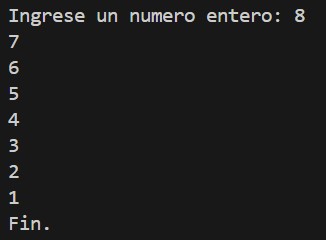
**PROPÓSITO DE LA TAREA**

Reutilizar el código mediante funciones recursivas para una programación óptima.

## Parte I: Funciones recursivas

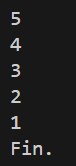
1. Implementar una función recursiva que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

<https://github.com/FelipeZapata137/Algoritmos-/blob/main/Parcial%203/D10E01.py>



1. Implementar una función recursiva que imprima en forma descendente de 5 a 1 de uno en uno.

<https://github.com/FelipeZapata137/Algoritmos-/blob/main/Parcial%203/D10E02.py>



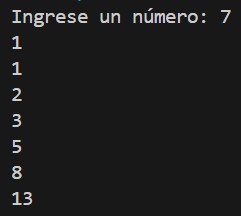
1. Usando funciones recursivas, calcular el factorial de un número n!.

El factorial de un número n se define como la multiplicación de todos sus números predecesores hasta llegar a uno. Por lo tanto 5!, leído como cinco factorial, sería 5\*4\*3\*2\*1.

<https://github.com/FelipeZapata137/Algoritmos-/blob/main/Parcial%203/D10E03.py>



1. Usando funciones recursivas, calcular la serie de fibonacci. Dicha serie calcula el elemento n sumando los dos anteriores n-1 + n-2. Se asume que los dos primeros elementos son 0 y 1. <https://github.com/FelipeZapata137/Algoritmos-/blob/main/Parcial%203/D10E04.py>



1. Implementar un método recursivo para ordenar los elementos de una lista.

<https://github.com/FelipeZapata137/Algoritmos-/blob/main/Parcial%203/D10E05.py>



## Parte II: Consulta de Algoritmos

Consultar y realizar una presentación sobre los siguientes algoritmos:

* Algoritmos de ordenamiento (inserción sort)

|  |
| --- |
| def insertion\_sort(arr):  # Recorremos desde el segundo elemento hasta el final  for i in range(1, len(arr)):  # Guardamos el valor actual y el índice anterior  current\_value = arr[i]  j = i - 1    # Desplazamos los elementos que sean mayores al valor actual  while j >= 0 and arr[j] > current\_value:  arr[j + 1] = arr[j]  j -= 1    # Insertamos el valor actual en su posición correcta  arr[j + 1] = current\_value  return arr  # Ejemplo de uso  numeros = [5, 3, 8, 6, 2]  print("Arreglo original:", numeros)  print("Arreglo ordenado:", insertion\_sort(numeros)) |

* Algoritmos de ordenamiento (quick sort)

|  |
| --- |
| def quick\_sort(array):  # Caso base: Si el array tiene 0 o 1 elemento, ya está ordenado  if len(array) <= 1:  return array  # Elegir un pivote (normalmente el último elemento)  pivot = array[-1]  # Dividir en listas menores y mayores al pivote  less\_than\_pivot = [x for x in array[:-1] if x <= pivot]  greater\_than\_pivot = [x for x in array[:-1] if x > pivot]  # Ordenar recursivamente y combinar  return quick\_sort(less\_than\_pivot) + [pivot] + quick\_sort(greater\_than\_pivot)  # Ejemplo de uso  if \_\_name\_\_ == "\_\_main\_\_":  unsorted\_list = [10, 7, 8, 1, 6, 3, 5]  sorted\_list = quick\_sort(unsorted\_list)  print("Lista ordenada:", sorted\_list) |