ESTRUCTURA DE DATOS FUNDAMENTALES Y ALGORITMOS GUÍA DE LABORATORIO 03 – 04
Recursividad – Arreglos y Matrices

Asignatura	Alumnos	Fecha y Firma
Algoritmos y solución de problemas	Ramos Lopez Isaias [27202506]	Sunday, 14 January 2024
	Yanasupo Romero Thayli Roxana [27210117]	

Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

1. Objetivos:

- Escribir algoritmos y codificar haciendo uso de recursividad, arreglos y matrices.

2. Equipos, herramientas o materiales

- Computador, Software: Python, Algoritmos

3. Fundamento teórico

3.1. Conceptos Clave

3.1.1. RECURSIVIDAD

La recursividad es un concepto matemático y computacional que se refiere a la capacidad de una función o algoritmo para llamarse a sí mismo. Esto puede parecer contradictorio, pero en realidad es una herramienta muy poderosa que se puede utilizar para resolver una amplia gama de problemas.

Para entender la recursividad en Python, es importante comprender los conceptos básicos de la recursión. Hay dos partes principales de una función recursiva:

- **El caso base:** el caso base es una condición que le indica a la función que debe dejar de llamarse a sí misma y devolver un valor.
- **El caso recursivo**: el caso recursivo es la condición que le indica a la función que debe llamarse a sí misma con diferentes argumentos.

Ejemplo 01:

Aquí hay un ejemplo de una función recursiva en Python que calcula la factorial de un número:

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

Esta función funciona de la siguiente manera:

El caso base es n == 0. En este caso, la función simplemente devuelve 1.

El caso recursivo es n > 0. En este caso, la función calcula la factorial de n - 1 y lo multiplica por n.

Ejemplo 03:

El cálculo del Fibonacci: se puede utilizar la recursividad para calcular los números de Fibonacci. Los números de Fibonacci son una secuencia de números en la que cada número es la suma de los dos números anteriores. La siguiente función calcula los primeros n números de Fibonacci:

```python
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)


if __name__ == "__main__":
    altura = int(input("Introduzca la cantidad de números de Fibonacci que desea imprimir: "))
    for i in range(altura):
        print(fibonacci(i))
```

Esta implementación funciona de la siguiente manera:

En el caso base, si n es igual a 0 o 1, la función simplemente devuelve n.
En el caso recursivo, la función hace lo siguiente:
Calcula el siguiente número de Fibonacci.
Devuelve la suma de los dos últimos números de Fibonacci.

3.1.2. ARREGLOS y MATRICES

Para trabajar con matrices en Python 3.11, tienes dos opciones principales:

- **Usar el módulo numpy:**

Instala el módulo numpy si aún no lo tienes: pip install numpy
Importa el módulo numpy en tu código:

```python
import numpy as np
```

Crea las matrices usando la función np.array():

```python
matriz_1 = np.array([[1, 2, 3], [4, 5, 6]])
matriz_2 = np.array([[7, 8, 9], [10, 11, 12]])
```

NumPy arrays: son una estructura de datos de Python que se utilizan para almacenar datos numéricos. Las matrices de NumPy son matrices multidimensionales, lo que significa que pueden tener más de una dimensión.

- **Creación de matrices**

Las matrices de NumPy se pueden crear de varias maneras. Una forma es usar la función np.array(). La función np.array() toma una secuencia de datos como entrada y crea una matriz de NumPy de la misma forma.

Por ejemplo, el siguiente código crea una matriz de NumPy de números enteros:

```python
import numpy as np

matriz = np.array([1, 2, 3])
```

Este código crea una matriz de NumPy de una dimensión con tres elementos.

Acceso a elementos de matrices

Los elementos de las matrices de NumPy se pueden acceder mediante su índice. El índice de una matriz de NumPy es un número que identifica a un elemento específico de la matriz.

Por ejemplo, el siguiente código imprime el primer elemento de la matriz matriz:

```python
print(matriz[0])
```

Operaciones con matrices

Las matrices de NumPy se pueden manipular mediante operaciones matemáticas. Las operaciones matemáticas comunes que se pueden realizar con matrices incluyen suma, resta, multiplicación y división.

Por ejemplo, el siguiente código suma dos matrices:

```python
matriz_1 = np.array([[1, 2, 3], [4, 5, 6]])
matriz_2 = np.array([[7, 8, 9], [10, 11, 12]])

matriz_suma = matriz_1 + matriz_2

print(matriz_suma)
```

Funciones de NumPy para matrices

NumPy proporciona muchas funciones útiles para el análisis y la manipulación de matrices. Algunas de las funciones de NumPy más comunes para matrices incluyen:

np.shape(): Devuelve la forma de una matriz.
np.reshape(): Cambia la forma de una matriz.
np.transpose(): Transpone una matriz.
np.dot(): Multiplica dos matrices.
np.sum(): Suma los elementos de una matriz.
np.mean(): Calcula la media de los elementos de una matriz.
np.std(): Calcula la desviación estándar de los elementos de una matriz.

4. Desarrollo y Actividades

Ejercicio parte 01:

Recursividad:

1) Ejercicio 1: Escribe una función recursiva que imprima los números pares del 1 al 100.

```python
# 01: Escribe una función recursiva que imprima los números pares del 1 al 100.
# Se define la función print_even_numbers usando "type hints"
def print_even_numbers(n: int = 1) -> None:
    """
    Esta función imprime los números pares de 1 a 100

    Parameters:
        n (int, optional): limite superior

    Returns:
        None

    """
    if n == 100:
        # caso base: se detiene si es 100
        return
    if n % 2 == 0:
        # imprime los números pares de 1 a 100
        print(n, end=" - ")
    # caso recursivo: llama a la función pasando como argumento el siguiente número
    print_even_numbers(n + 1)

# llamada de la función
print_even_numbers()
```

Ejecución

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
2 - 4 - 6 - 8 - 10 - 12 - 14 - 16 - 18 - 20 - 22 - 24 - 26 - 28 - 30 - 32 - 34 - 36 - 38 - 40 - 42 - 44 - 46 - 48
- 50 - 52 - 54 - 56 - 58 - 60 - 62 - 64 - 66 - 68 - 70 - 72 - 74 - 76 - 78 - 80 - 82 - 84 - 86 - 88 - 90 - 92 - 94
 - 96 - 98 - ⏎

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

2) Ejercicio 2: Escribe una función recursiva que imprima la suma de los números del 1 al n.

```python
# ------------------------------------------------------------------------------
# 02: Escribe una función recursiva que imprima la suma de los números del 1 al n
def recursive_sum(current_number: int, target_number: int, current_sum: int = 0) -> None:
    """
    Imprime la suma de los números del 1 al n

    Parameters:
        current_number (int): número actual
        target_number (int): número final
        current_sum (int, optional): suma actual

    Returns:
        None
    """
    if current_number > target_number:
        # caso base, termina con la ejecución
        return
    # almacena en current_sum la suma de current_sum + current_number
    current_sum: int =current_sum + current_number
    # imprime la suma actual
    print(current_sum)
    # caso recursivo: llama a la función
    recursive_sum(current_number + 1, target_number, current_sum)

# llamada de la función
recursive_sum(1, 5)
```

Ejecución

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
1
3
6
10
15

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
>< ⊗ 0 ⚠ 0  0
```

3) Ejercicio 3: Escribe una función recursiva que imprima la pirámide de números del 1 al n.

```python
# ------------------------------------------------------------------------------
# 03: Escribe una función recursiva que imprima la pirámide de números del 1 al n
def print_number_pyramid(n: int, current_row: int = 1) -> None:
    """
    Imprime la pirámide de números del 1 al n

    Parameters:
        n (int): número de filas de la pirámide
        current_row (int, optional): fila actual de la pirámide

    Returns:
        None

    """
    if current_row > n:
        # caso base: Se detiene si n es 100
        return

    # imprime espacios antes de los números de la fila actual
    print(" " * (n - current_row), end="")

    # imprime los números de la fila actual de manera ascendente
    for num in range(1, current_row + 1):
        print(num, end="")
    # imprime los números de la fila actual de manera descendente
    for num in range(current_row - 1, 0, -1):
        print(num, end="")

    # pasar a la siguiente línea después de imprimir la fila.
    print()

    # caso recursivo: imprime recursivamente la siguiente fila
    print_number_pyramid(n, current_row + 1)

# llamada de la función
print_number_pyramid(9)
```

Ejecución

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
        1
       121
      12321
     1234321
    123454321
   12345654321
  1234567654321
 123456787654321
12345678987654321

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
>< ⊗ 0 ⚠ 0  0
```

4) Ejercicio 4: Escribe una función recursiva que imprima la pirámide de números invertidos del 1 al n.

```python
# ------------------------------------------------------------------------------
# 04: Escribe una función recursiva que imprima la pirámide de números invertidos del 1 al n
def print_inverted_number_pyramid(n: int, current_row: int = 1) -> None:
    """
    Imprime la pirámide de números invertidos del 1 al n.

    Parameters:
        n (int): número de filas de la pirámide
        current_row (int, optional): la fila actual que se está imprimiendo

    Returns:
        None
    """
    if current_row > n:
        # caso base: Se detiene si current_row es mayor a n
        return

    # imprime espacios antes de los números invertidos de la fila actual
    print(" " * (n - current_row), end="")

    # imprime los números de la fila actual de manera descendente
    for num in range(current_row, 0, -1):
        print(num, end="")

    # imprime los números de la fila actual de manera ascendente
    for num in range(2, current_row + 1):
        print(num, end="")

    # pasar a la siguiente línea después de imprimir la fila.
    print()

    # caso recursivo: imprime recursivamente la siguiente fila
    print_inverted_number_pyramid(n, current_row + 1)

# llamada de la función
print_inverted_number_pyramid(6)
```

Ejecución

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
     1
    212
   32123
  4321234
 543212345
65432123456

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

5) Ejercicio 2: Escribe una función recursiva que imprima la tabla de multiplicar del n.

```python
# ------------------------------------------------------------------------------
# 05:  Escribe una función recursiva que imprima la tabla de multiplicar del n.
def print_multiplication_table(n: int, multiplier: int = 1) -> None:
    """
    Imprime la tabla de multiplicar del n.

    Parameters:
        n (int): el número para el cual se imprimirá la tabla de multiplicar
        multiplier (int, optional): el multiplicador de la fila de la tabla

    Returns:
        None

    """
    if multiplier > 10:
        # caso base: esta instrucción if verifica si el multiplicador es mayor que 10.
        return
    # esta variable almacena el producto de n y el multiplicador actual
    result: int = n * multiplier
    # el resultado (entero) se imprime usando la sintaxis de format (f), \t: carácter de tabulación
    print(f"\t{n} x {multiplier} = {result}")
    # caso recursivo: esta función se llama de forma recursiva para imprimir la siguiente fila
    print_multiplication_table(n, multiplier + 1)

# llamada de la función
print_multiplication_table(5)
```

Ejecución

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
        5 x 1 = 5
        5 x 2 = 10
        5 x 3 = 15
        5 x 4 = 20
        5 x 5 = 25
        5 x 6 = 30
        5 x 7 = 35
        5 x 8 = 40
        5 x 9 = 45
        5 x 10 = 50

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

Arreglos y Matrices:

Para los siguientes ejercicios estas librerías usaremos, numpy y lo asignaremos como np además de importar la clase Tuple del módulo typing, esto nos ayudará para hacer type hints.

```python
import numpy as np
from typing import Tuple
```

6) Crea una matriz de números reales.

```python
# ------------------------------------------------------------------------------
# 06: Crea una matriz de números reales
# usando types hints (np.ndarray para matrizes)
def real_matrix() -> np.ndarray:
    """
    Crea una matriz de números reales.

    Returns:
        np.ndarray: matriz de números reales.
    """
    return np.array([
        [7, 2, 5],
        [4, 5, 8],
        [6, 0, 9]
    ])

# llamada de la función
print(real_matrix())
```

Ejecución

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS                fish

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
[[7 2 5]
 [4 5 8]
 [6 0 9]]

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

7) Crea una matriz de números complejos.
8) Crea una matriz de matrices.

```python
# ------------------------------------------------------------------------------------------
# 07: Crea una matriz de números complejos
def complex_matrix() -> np.ndarray:
    """
    Crea una matriz de números complejos.

    Returns:
        np.ndarray: matriz de números complejos.
    """
    return np.array([
        [1 + 2j, 3 + 4j, 6 + 5j],
        [5 + 6j, 7 + 8j, 0],
        [9 + 1j, 2, 2j]
    ], dtype=complex)

# llamada de la función
print(complex_matrix())


# ------------------------------------------------------------------------------------------
# 08: Crea una matriz de matrices
def matrix_matrix() -> np.ndarray:
    return np.array([
        [8, 9, 3, 5],
        [9, 4, 0, 2],
        [0, 5, 2, 1],
        [7, 8, 9, 0]
    ])

# llamada de la función
print(matrix_matrix())
```

Ejecución

La ejecución de la pregunta 7, está al lado izquierdo, mientras que la 8 en el lado derecho.

Implementamos una función lambda que son parecidas a las funciones flecha en javascript, que nos servirá para crear una matriz.

```python
# Función lambda para crear matrices aleatorias, recibe 2 parametros, el número de filas y el número de columnas.
random_matrix_generator: np.ndarray = lambda rows, cols: np.random.randint(1, 10, size=(rows, cols))
```

9) Accede al elemento central de una matriz.

```python
# ------------------------------------------------------------------------------
# 09: Accede al elemento central de una matriz
def central_matrix(matrix: np.ndarray) -> np.ndarray:
    """
    Accede al elemento central de una matriz

    Parameters:
        matrix (np.ndarray): Matriz

    Returns:
        float: Elemento central de la matriz
    """
    if matrix.shape[0] != matrix.shape[1]:
        # Verifica que sea una matriz cuadrada
        return 'matriz de dimensiones incorrectas'
    return matrix[int(matrix.shape[0] / 2)][int(matrix.shape[1] / 2)]

# creando la matriz A con la función random_matrix_generator
A: np.ndarray = random_matrix_generator(5, 5)
# la matriz A
print(f"La matriz A es:\n{A}\n")
# llamada de la función
num_central: int = central_matrix(A)
print(f'El elemento central de la matriz A es: {num_central}')
```

Ejecución

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
La matriz A es:
[[4 7 4 4 1]
 [7 7 8 1 6]
 [1 9 1 5 2]
 [4 2 2 1 9]
 [4 2 2 2 5]]

El elemento central de la matriz A es: 1

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

10) Suma dos matrices de diferentes tamaños.

```python
# ------------------------------------------------------------------------------
# 10:  Suma dos matrices de diferentes tamaños.
def matrix_sum(A: np.ndarray, B: np.ndarray) -> np.ndarray:
    """
    Suma dos matrices de diferentes tamaños.

    Parameters:
        A (np.ndarray): La primera matriz
        B (np.ndarray): La segunda amtriz

    Returns:
        np.ndarray: La suma de las matrices
    """
    # Usamos concatenate() de numpy para concatenar matrices de diferentes tamaños.
    result_matrix: np.ndarray = A + B
    return result_matrix

# creando la matriz A, B con la función random_matrix_generator
A: np.ndarray = random_matrix_generator(3, 3)
# la matriz A
print(f"La matriz A es:\n{A}\n")
# la matriz B
B: np.ndarray = random_matrix_generator(3, 3)
print(f"La matriz B es:\n{B}\n")
# llamada de la función, pasandole como arguemnto la matriz A y la matriz B
sum_matrix: np.ndarray = matrix_sum(A, B)
print(f'Suma de las matrices es:\n{sum_matrix}')
```

Ejecución

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
La matriz A es:
[[8 8 9]
 [8 4 7]
 [3 1 4]]

La matriz B es:
[[8 1 2]
 [3 3 2]
 [9 6 6]]

Suma de las matrices es:
[[16  9 11]
 [11  7  9]
 [12  7 10]]

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

11) Multiplica una matriz por un número.
12) Calcula la media de los elementos de una matriz.

```python
# ------------------------------------------------------------------------------------
# 11: Multiplica una matriz por un número.
def multiply_matrix_by_scalar(matrix: np.ndarray, scalar: float) -> np.ndarray:
    """
    Multiplica una matriz por un número.

    Parameters:
        matrix (np.ndarray): Matriz
        scalar (float): Número

    Returns:
        np.ndarray: Matriz multiplicada por el número
    """
    return matrix * scalar

# creamos la matriz X, hacemos uso de la función multiply_matrix_by_scalar
X: np.ndarray = random_matrix_generator(3, 3)
print(f'la matriz X es:\n{X}\n')
scalar_dot_matrix: np.ndarray = multiply_matrix_by_scalar(X, 6)
print(f'La matriz multiplicada por 6 es:\n{scalar_dot_matrix}\n')


# ------------------------------------------------------------------------------------
# 12:  Calcula la media de los elementos de una matriz
def matrix_mean(matrix: np.ndarray) -> float:
    """
    Calcula la media de los elementos de una matriz

    Parameters:
        matrix (np.ndarray): Matriz

    Returns:
        float: Media
    """
    return np.mean(matrix)

# usamos la matriz X anteriormente creada, hacemos uso de la funcion matrix_mean
mean_matrix: float = matrix_mean(X)
print(mean_matrix)
```

Ejecución, la 11 y la 12.

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
la matriz X es:
[[2 5 1]
 [6 9 1]
 [1 7 6]]

La matriz multiplicada por 6 es:
[[12 30  6]
 [36 54  6]
 [ 6 42 36]]

La media de la matriz es: 4.222222222222222


UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

Ejercicio parte 01:

Ejercicio 1:

Crea una matriz de números aleatorios de tamaño 100x100.

```python
# ------------------------------------------------------------------------------
# 01: Crea una matriz de números aleatorios de tamaño 100x100
def random_matrix() -> np.ndarray:
    # crea una matriz de números aleatorios de tamaño 100 x 100
    return np.random.rand(100, 100)

# llamada de la función
matrix_random: np.ndarray = random_matrix()
print(f'matriz aleatoria de tamaño 100x100 es:\n{matrix_random}')
```

Ejecución

```
❯ python fn_exercises.py
matriz aleatoria de tamaño 100x100 es:
[[0.74200342 0.45198299 0.26641299 ... 0.41621582 0.57941297 0.47688648]
 [0.56976452 0.55787139 0.06001852 ... 0.76060781 0.72468454 0.46232393]
 [0.17202899 0.89070293 0.66239996 ... 0.87757622 0.55285889 0.07717421]
 ...
 [0.42133105 0.30035122 0.98094536 ... 0.48755531 0.03788061 0.55829355]
 [0.14550036 0.2439349  0.80252508 ... 0.65734353 0.13313658 0.51866012]
 [0.45455668 0.11872588 0.65234085 ... 0.62612624 0.53008709 0.60671178]]

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

Ejercicio 2:

Calcula la media, la mediana y la desviación estándar de los elementos de una matriz.

```python
# ------------------------------------------------------------------------------
# 02: Calcula la media, la mediana y la desviación estándar de los elementos de una matriz
def calculate_statistics(matrix: np.ndarray) -> Tuple[float, float, float]:
    """
    Calcula la media, mediana y desviación estándar de los elementos de una matriz

    Parameters:
        matrix (np.ndarray): Matriz

    Returns:
        Tuple[float, float, float]: Una tupla que contiene la media, la mediana y la desviación estándar
    """
    # calcula la media
    mean_value = np.mean(matrix)
    # llama la función matrix_mean para obtener la media de la matriz
    median_value = matrix_mean(matrix)
    # calcula la desviación estándar
    std_deviation = np.std(matrix)

    return mean_value, median_value, std_deviation

# creamos la matriz A, llamamos a la función calculate_statistics
A: np.ndarray = random_matrix_generator(5, 5)
print(f"La matriz A es:\n{A}\n")
# usando la asignación múltiple o desempaquetado de tuplas en Python tenemos:
mean_value, median_value, std_deviation = calculate_statistics(A)
print(mean_value, median_value, std_deviation)
```

Ejecución

```
❯ python fn_exercises.py
La matriz A es:
[[2 6 4 3 1]
 [3 3 2 6 5]
 [6 2 7 9 6]
 [6 2 3 7 2]
 [1 1 8 5 9]]

4.36 4.36 2.48
```

Ejercicio 3:

Escribe una función que encuentre el elemento máximo de una matriz.

```python
# ------------------------------------------------------------------------------
# 03: Escribe una función que encuentre el elemento máximo de una matriz
def find_max_element(matrix: np.ndarray) -> float:
    """
    Encuentra el elemento máximo de una matriz

    Parameters:
        matrix (np.ndarray): Matriz

    Returns:
        float: El elemento máximo de la matriz
    """
    # calcula el máximo elemento de la matriz
    return np.max(matrix)

# creamos la matriz B, llamamos a la función calculate_statistics
B: np.ndarray = random_matrix_generator(5, 5)
max_element: float = find_max_element(B)
print(f"Matriz B:\n{B}")
print(f"Elemento máximo: {max_element}")
```

Ejecución

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
Matriz B:
[[6 8 1 3 9]
 [4 8 5 3 5]
 [9 1 2 7 9]
 [1 7 4 9 5]
 [3 6 7 7 4]]
Elemento máximo: 9

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

Ejercicio 4:

Escribe una función que encuentre la submatriz de mayor suma de una matriz

```python
# ------------------------------------------------------------------------------
# 04: Escribe una función que encuentre la submatriz de mayor suma de una matriz
def find_submatrix_max(matrix: np.ndarray, count: int) -> [np.ndarray, int]:
    """
    Encuentra la submatriz de mayor suma de una matriz

    Parameters:
        matrix (np.ndarray): Matriz
        count (int): Número de elementos a obtener

    Returns:
        Tuple[np.ndarray, int]: Una tupla que contiene la submatriz de mayor suma y el valor de la suma
    """
    # convertirmos la matriz en un arreglo
    arr_numbers = np.array(matrix).flatten()

    if count <= 0:
        # Si count es 0 o negativo, retornar un arreglo vacío
        return []

    # Ordenar el arreglo en orden descendente
    sorted_arr = sorted(arr_numbers, reverse=True)
    # Obtener los primeros count elementos
    result_arr = sorted_arr[:count]

    # Obtener la suma
    sum_arr: int = sum(result_arr)

    return result_arr, sum_arr


# Creamos una matriz de 5 x 5 con random_matrix_generator
matrix: np.ndarray = random_matrix_generator(5, 5)
print(f"Matriz:\n{matrix}\n")
# usamos asignación múltiple para alamcenar los valores que nos retorna la función
submatrix, sum = find_submatrix_max(matrix, matrix.shape[0])
print(f"submatriz: {submatrix} y la suma: {sum}")
```

Ejecución

```
❯ python fn_exercises.py
Matriz:
[[6 5 7 2 3]
 [4 1 9 4 7]
 [6 7 7 1 5]
 [5 9 8 6 1]
 [8 8 1 9 9]]

submatriz: [9, 9, 9, 9, 8] y la suma: 44

UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```

.

Ejercicio 5:

Escribe una función que encuentre la matriz de covarianza de dos matrices.

```python
# ------------------------------------------------------------------------------
# 05: Escribe una función que encuentre la matriz de covarianza de dos matrices
def covariance_matrix(matrix_a: np.ndarray, matrix_b: np.ndarray) -> np.ndarray:

    if matrix_a.shape != matrix_b.shape:
        raise ValueError("Matrices must have the same shape for covariance calculation.")

    # Apilar las matrices para obtener una matriz combinada
    combined_matrix = np.vstack((matrix_a, matrix_b))

    # Calcular la matriz de covarianza
    # rowvar=False se utiliza para tratar cada columna como una variable y cada fila como una observación
    covariance_matrix = np.cov(combined_matrix, rowvar=False)

    return covariance_matrix

# creamos matrices: X e Y, llamamos a la función covariance_matrix
X: np.ndarray = random_matrix_generator(5, 5)
Y: np.ndarray = random_matrix_generator(5, 5)
matrix_covariance: np.ndarray = covariance_matrix(X, Y)
print(f"Matriz A:\n{X}\n")
print(f"Matriz B:\n{Y}\n")
print(f'Matriz de covarianza:\n{matrix_covariance}\n')
```

Ejecución

```
UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯ python fn_exercises.py
Matriz A:
[[6 4 5 9 8]
 [4 5 3 9 1]
 [5 6 4 3 3]
 [6 2 4 7 3]
 [3 4 1 3 2]]

Matriz B:
[[1 4 6 1 8]
 [2 5 9 5 9]
 [3 6 2 8 4]
 [3 6 5 7 8]
 [8 8 3 3 6]]

Matriz de covarianza:
[[ 4.54444444  0.77777778 -1.57777778  1.16666667 -1.35555556]
 [ 0.77777778  2.66666667 -0.44444444 -0.77777778  0.66666667]
 [-1.57777778 -0.44444444  5.06666667 -0.55555556  5.06666667]
 [ 1.16666667 -0.77777778 -0.55555556  8.27777778 -1.11111111]
 [-1.35555556  0.66666667  5.06666667 -1.11111111  8.62222222]]


UNSCH/ES-281/L-02 via 🐍 v3.11.6 (env)
❯
```