



*Practica: 04*  
*Nombre de la Practica: Ejercicios de*  
*Pypass*

Instituto Politécnico Nacional.  
Escuela Superior de Cómputo.  
Licenciatura en ciencia de datos.

Nombre de la materia: Desarrollo de Aplicaciones para el Análisis de Datos  
Grupo: 4AV1  
Profesora: Sandra Luz Morales Guitron

Lopez Mendez Emiliano

## INDICE

INTRODUCCION .....	3
DESAROLLO .....	3
FUNCIONES .....	3
OOP.....	15
EXCEPTIONS .....	30
CONCLUSIONES.....	32

## INTRODUCCION

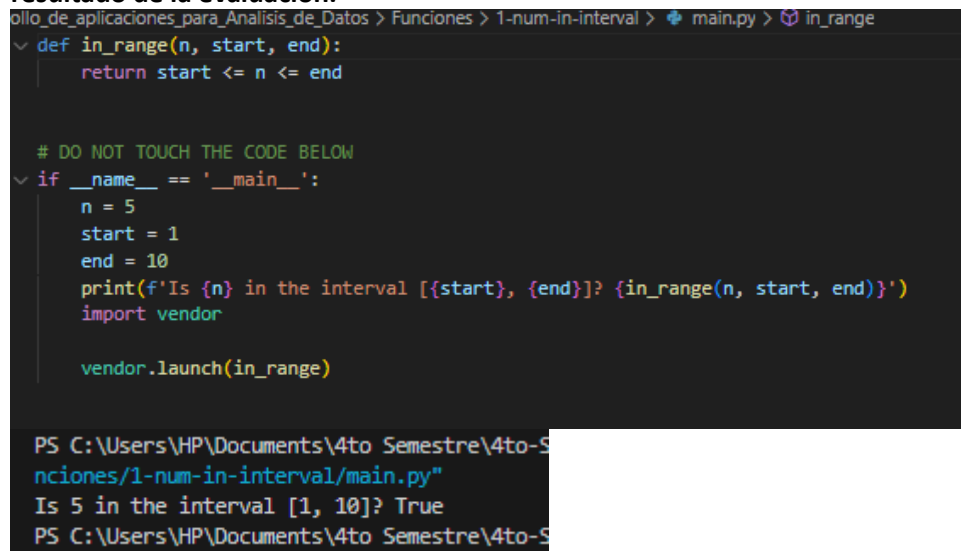
En esta practica se llevo la instalación de un Editor de programación para poder programar con la versión mas actualizada de Python. Escogiendo la computadora que vamos a usar, el sistema operativa y el Editor que se usaran.

## DESAROLLO

Se llevo acabo el análisis y configuración de varios códigos ya adentro de pypass. Se modificaron para poder imprimir los resultados deseados y también para poder imprimir dichos programas se hizo un archivo aparte llamado args.py con las variables que se piden del main.

### FUNCIONES

1. **num-in-interval** Este código define una función llamada `in_range` que verifica si un número `n` se encuentra dentro del intervalo cerrado definido por `start` y `end`. Devuelve `True` si el número está dentro del rango, y `False` en caso contrario. En la ejecución, verifica si el número 5 está dentro del intervalo `[1, 10]`, imprimiendo el resultado de la evaluación.



```
oio_de_aplicaciones_para_Analisis_de_Datos > Funciones > 1-num-in-interval > main.py > in_range
def in_range(n, start, end):
    return start <= n <= end

# DO NOT TOUCH THE CODE BELOW
if __name__ == '__main__':
    n = 5
    start = 1
    end = 10
    print(f'Is {n} in the interval [{start}, {end}]? {in_range(n, start, end)}')
    import vendor

    vendor.launch(in_range)

PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Aplicaciones\1-num-in-interval\main.py
Is 5 in the interval [1, 10]? True
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Aplicaciones\1-num-in-interval\main.py
```

2. **extract-evens** Este código define la función `extract_evens` que utiliza una **\*\*comprensión de listas\*\*** para extraer los números pares de una lista dada. En la ejecución, toma la lista `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` y extrae los números pares, imprimiendo el resultado: `[2, 4, 6, 8, 10]`.

```

def extract_evens(numbers):
    return [number for number in numbers if number % 2 == 0]

# DO NOT TOUCH THE CODE BELOW
if __name__ == '__main__':
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print(f'Original list: {numbers}')
    print(f'Even numbers: {extract_evens(numbers)}')
    import vendor

    vendor.launch(extract_evens)

PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Datos> cd .\Funciones\2-extract-evens\main.py
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even numbers: [2, 4, 6, 8, 10]
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Datos>

```

3. split-case El código define la función `split\_case`, que separa una lista de palabras en dos listas: una con palabras en **mayúsculas** y otra con palabras en **minúsculas**.

- Si una palabra contiene tanto mayúsculas como minúsculas, se ignora.
- En la ejecución, con la lista `['cocodrilo', 'ZEBRA', 'MAPACHE', 'Serpiente', 'ballena']`, el resultado es dos listas:
  - Minúsculas: `['cocodrilo', 'ballena']`
  - Mayúsculas: `['ZEBRA', 'MAPACHE']`.

Se devuelve una tupla con ambas listas.

```

def split_case(s):
    upper = []
    lower = []
    for i in s:
        if any(j.isupper() for j in i) and any(j.islower() for j in i):
            pass
        elif i.isupper():
            upper.append(i)
        else:
            lower.append(i)
    result = (lower, upper)
    return result

# DO NOT TOUCH THE CODE BELOW
if __name__ == '__main__':
    lista = ['cocodrilo', 'ZEBRA', 'MAPACHE', 'Serpiente', 'ballena']
    print(f'Original list: {lista}')
    print(f'Split case: {split_case(lista)}')
    import vendor

    vendor.launch(split_case)

PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Datos> cd .\Funciones\3-split-case\main.py
Original list: ['cocodrilo', 'ZEBRA', 'MAPACHE', 'Serpiente', 'ballena']
Split case: (['cocodrilo', 'ballena'], ['ZEBRA', 'MAPACHE'])
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Datos>

```

4. Perfect El código verifica si un número es perfecto sumando sus divisores propios y comparándolos con el número dado.

```
rollo_de_aplicaciones_para_Analisis_de_Datos > Funciones > 4-  
def is_perfect(n: int) -> bool:  
    if n < 1:  
        return False  
    divisors = [1]  
    for i in range(2, n):  
        if n % i == 0:  
            divisors.append(i)  
    return sum(divisors) == n  
  
# DO NOT TOUCH THE CODE BELOW  
if __name__ == '__main__':  
    print(is_perfect(28))  
    import vendor  
  
    vendor.launch(is_perfect)
```

PS C:\Users\HP\Documents\Funciones\4-perfecto> python 4-perfecto.py  
True  
PS C:\Users\HP\Documents\Funciones\4-perfecto>

5. Palindrome El código verifica si una palabra o frase es un palíndromo, ignorando mayúsculas y espacios, comparando el texto con su versión invertida.

```

You, 6 minutes ago | 2 authors (unmetro and one other)
1 def is_palindrome(word):
2     word = word.lower()
3     word = word.replace(" ", "")
4     return word == word[::-1]
5
6
7
8 # DO NOT TOUCH THE CODE BELOW
9 if __name__ == '__main__':
10     print(is_palindrome("Never odd or even"))
11     import vendor
12
13     vendor.launch(is_palindrome)
14
True
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Computaciones\5-palindrome\main.py"
True
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Computaciones\5-palindrome\main.py"

```

6. **count-vowels-recursive** Este código define una función recursiva llamada `count_vowels` que cuenta el número de vocales (incluyendo tildadas) en una cadena de texto dada. La función revisa cada carácter de la cadena: si es una vocal, suma 1 y continúa con el resto del texto; si no, simplemente avanza. Cuando la cadena está vacía, la recursión termina y devuelve el total de vocales encontradas.

```

unmetro, 11 minutes ago | 1 author (unmetro)
1 def count_vowels(text):
2     # Lista de vocales incluyendo las con tilde
3     vowels = "aeiouáéíóúáEIOUÁÉÍÓÚ"
4
5     # Caso base: Si la cadena está vacía, no hay más vocales
6     if text == "":
7         return 0
8
9     # Si el primer carácter es una vocal, sumamos 1 y seguimos con el resto de la cadena
10    if text[0] in vowels:
11        return 1 + count_vowels(text[1:])
12
13    # Si no es una vocal, seguimos con el resto de la cadena sin sumar
14    else:
15        return count_vowels(text[1:])
16
17
18 #ejecucion
19 texto = "Ayer empecé a programar"
20 resultado = count_vowels(texto)
21 print(f"El texto '{texto}' tiene {resultado} vocales.")
22
True
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Computaciones\6-count-vowels-recursive.py"
El texto 'Ayer empecé a programar' tiene 9 vocales.
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Computaciones\6-count-vowels-recursive.py"

```

7. **Pangram** Este código verifica si un texto es un pangrama, es decir, si contiene todas las letras del alfabeto al menos una vez. Primero convierte el texto a minúsculas y elimina los caracteres no alfabéticos. Luego compara las letras presentes en el texto con un

conjunto del alfabeto. Si todas las letras del alfabeto están en el texto, devuelve `True`; de lo contrario, `False`.

```

import string
unmetro, 12 minutos ago • tu tia wey

def is_pangram(text):
    # Convertimos el texto a minúsculas para evitar diferencias de mayúsculas/minúsculas
    text = text.lower()

    # Eliminamos espacios y caracteres no alfabéticos
    text = ''.join(char for char in text if char.isalpha())

    # Creamos un conjunto con todas las letras del abecedario
    alphabet = set(string.ascii_lowercase)

    # Comparamos si el conjunto del alfabeto está contenido en el texto dado
    return alphabet.issubset(set(text))

# ejecucion
frase1 = "Un jugoso zumo de piña y kiwi bien frío es exquisito y no lleva alcohol"
frase2 = "No utilizo todas las letras del abecedario"

print(f"¿La frase 1 es pangrama? {is_pangram(frase1)}")
print(f"¿La frase 2 es pangrama? {is_pangram(frase2)}")

```

nciones/6-count-vowels-recursive.py"

El texto 'Ayer empecé a programar' tiene 9 vo

PS C:\Users\HP\Documents\4to Semestre\4to-Sem

nciones/7-pangram.py"

¿La frase 1 es pangrama? True

¿La frase 2 es pangrama? False

PS C:\Users\HP\Documents\4to Semestre\4to-Sem

8. cycle-alphabet Este código genera caracteres del alfabeto de forma cíclica. Utiliza un generador para iterar sobre el alfabeto en minúsculas repetidamente. Para cada carácter, utiliza el operador módulo (`%`) para asegurarse de que la posición se reinicie al alcanzar el final del alfabeto. En los ejemplos de ejecución, genera secuencias de 10 y 43 caracteres en ciclo.

```

1  import string          unmetro, 12 minutes ago • tu tia wey
2
3
4  def cycle_alphabet(num_chars):
5      # Obtenemos el alfabeto en minúsculas
6      alphabet = string.ascii_lowercase
7      alphabet_length = len(alphabet)
8
9      # Generamos caracteres de forma cíclica
10     for i in range(num_chars):
11         yield alphabet[i % alphabet_length]
12
13
14     # ejecucion
15     num_chars_10 = 10
16     resultado_10 = ''.join(cycle_alphabet(num_chars_10))
17     print(f"Con num_chars= {num_chars_10} : {resultado_10}")
18
19     num_chars_43 = 43
20     resultado_43 = ''.join(cycle_alphabet(num_chars_43))
21     print(f"Con num_chars= {num_chars_43} : {resultado_43}")

```

nciones/8-cycle-alphabet.py"

Con num\_chars= 10 : abcdefghij

Con num\_chars= 43 : abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Datos>

9. bubble-sort Este código implementa el **algoritmo de ordenamiento por burbuja**. Crea una copia de la lista original y recorre repetidamente la lista, comparando elementos adyacentes e intercambiándolos si están en el orden incorrecto. De esta forma, los elementos más grandes "burbujearán" hacia el final en cada pasada. Al finalizar, retorna la lista ordenada. En la ejecución, muestra una lista desordenada y luego la lista ordenada con este método.



```
1 def bsort(input_list):  
2     # copia de lista original  
3     lst = input_list[:]  unmetro, 12 minutes ago • tu tia wey  
4     n = len(lst)  
5  
6     # Implementamos el algoritmo de burbuja  
7     for i in range(n):  
8         # En cada pasada, los elementos más grandes se mueven hacia el final  
9         for j in range(0, n - i - 1):  
10             # Intercambiamos si el elemento actual es mayor que el siguiente  
11             if lst[j] > lst[j + 1]:  
12                 lst[j], lst[j + 1] = lst[j + 1], lst[j]  
13  
14  
15     return lst  
16  
17  
18 # ejecucion  
19 lista = [64, 34, 25, 12, 22, 11, 90]  
20 lista_ordenada = bsort(lista)  
21  
22 print(f"Lista original: {lista}")  
23 print(f"Lista ordenada: {lista_ordenada}")
```

Con num\_chars= 43 : abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencias\9-bubble-sort.py  
Lista original: [64, 34, 25, 12, 22, 11, 90]  
Lista ordenada: [11, 12, 22, 25, 34, 64, 90]  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencias\9-bubble-sort.py

10. consecutive-seq Este código implementa una función recursiva que busca un número que aparezca consecutivamente una cantidad específica de veces en una lista. Si encuentra el número que cumple con las repeticiones requeridas, lo devuelve; de lo contrario, continúa recorriendo la lista. Si no se encuentra ninguna coincidencia al llegar al final, devuelve `None`. En la ejecución, busca un número que se repita tres veces consecutivas en la lista dada.

```
unmetro, 12 minutos ago | autor (unmetro)
def consecutive_seq(values, repetitions, index=0, count=1):
    # Caso base: Si llegamos al final de la lista y no encontramos coincidencias
    if index >= len(values) - 1:
        return None

    # Si el elemento actual es igual al siguiente, aumentamos el contador
    if values[index] == values[index + 1]:
        count += 1
    else:
        # Si no son iguales, reiniciamos el contador
        count = 1

    # Si el contador alcanza las repeticiones requeridas, devolvemos el número
    if count == repetitions:
        return values[index]

    # Llamada recursiva al siguiente elemento
    return consecutive_seq(values, repetitions, index + 1, count)

# ejecucion
valores = [1, 74, 56, 56, 56, 332, 8, 8, 8]
repeticiones = 3

resultado = consecutive_seq(valores, repeticiones)
print(f"Resultado: {resultado}")

PS C:\Users\HP\Documents\4to S
nciones/10-consecutive-seq.py
Resultado: 56
PS C:\Users\HP\Documents\4to S
```

11. magic-square Este código verifica si una matriz cuadrada es un cuadrado mágico, es decir, si las sumas de sus filas, columnas y diagonales son iguales.

```

unmetro, 12 minutos ago | 1 author (unmetro)
def is_magic_square(matrix):
    # Obtener la suma de la primera fila como referencia
    n = len(matrix)
    sum_ref = sum(matrix[0])

    # Comprobar las sumas de las filas
    for row in matrix:
        if sum(row) != sum_ref:
            return False

    # Comprobar las sumas de las columnas
    for col in range(n):
        if sum(matrix[row][col] for row in range(n)) != sum_ref:
            return False

    # Comprobar la suma de la diagonal principal
    if sum(matrix[i][i] for i in range(n)) != sum_ref:
        return False

    # Comprobar la suma de la diagonal secundaria
    if sum(matrix[i][n - i - 1] for i in range(n)) != sum_ref:
        return False

    return True

# Ejemplo de uso:
matrix = [
    [4, 9, 2],
    [3, 5, 7],
    [8, 1, 6]
]

print(is_magic_square(matrix))

# Escribe una función que, dada una lista de listas (como matriz cuadrada), determina si es o
# no un cuadrado mágico. Esto a través de la suma de columnas, filas y diagonal

```

PS C:\Users\HP\Documents\4to Semestre\Funciones\11-magic-square/main.py"
True
PS C:\Users\HP\Documents\4to Semestre\Funciones\11-magic-square/main.py"

12. nested-add Este código calcula la suma total de una lista que puede contener otras listas anidadas mediante una función recursiva.

```

unmetro, 12 minutos ago | 1 author (unmetro)
1 def add(lst):
2     total = 0
3     for element in lst:
4         if isinstance(element, list): # Si el elemento es una lista, llama a la función recursivamente
5             total += add(element)
6         else:
7             total += element # Si es un número, lo suma al total
8     return total
9
10 lst = [1, [2, 4], 5, [6, [7, 8]]]
11 print(f"Suma de la lista: {add(lst)}")
12
13 #Regresa la suma total de una lista que puede tener listas anidadas a traves de una funicon recursiva

```

Funciones/12-nested-add/main.py
True
PS C:\Users\HP\Documents\4to Semestre\Funciones\12-nested-add/main.py"
Suma de la lista: 33
PS C:\Users\HP\Documents\4to Semestre\Funciones\12-nested-add/main.py"

13. fibonacci-recursive Este código calcula el enésimo número de Fibonacci mediante una función recursiva, actualizando los valores de los dos términos previos en cada llamada hasta alcanzar la posición deseada.

```
Desarrollo_de_aplicaciones_para_Análisis_de_Datos > Funciones > 13-fibonacci-recursive > main.py > fibonacci

1 def fibonacci(n: int, a: int, b: int, count: int) -> int:
2     a, b = b, a + b # Actualizar los valores de 'a' y 'b'
3     count += 1 # Incrementar el contador
4
5     if count == n + 1:
6         return b
7     else:
8         return fibonacci(n, a, b, count) # Retornar el valor de la llamada recursiva
9
10 count = 2
11 n = 10
12 print(f"El valor n-esimo fibonacci es: {fibonacci(n, a=0, b=1, count=count)}")
13
```

```
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Desarrollo_de_aplicaciones_para_Análisis_de_Datos\Funciones\13-fibonacci-recursive> python main.py
El valor n-esimo fibonacci es: 55
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Desarrollo_de_aplicaciones_para_Análisis_de_Datos\Funciones\13-fibonacci-recursive>
```

14. Hyperfactorial Este código calcula el hiperfactorial de un número de forma recursiva, sumando en cada llamada el resultado de elevar el número actual a sí mismo hasta llegar a 0.

```
Desarrollo_de_aplicaciones_para_Análisis_de_Datos > Funciones > 14-hyperfactorial > main.py > hyperfactorial

1 def hyperfactorial(n: int, sum: int) -> int:
2     if n == 0:
3         return sum
4     else:
5         h_factorial = pow(n, n)
6         sum += h_factorial
7         return hyperfactorial(n - 1, sum)
8
9 n = 5
10 sum = 0
11 print(f"El hiperfactorial de {n} es: {hyperfactorial(n, sum)}")
12
13 # El hiperfactorial de un numero n se calcula como el producto sucesivo de los decrementos de
14 # n elevado a si mismo. Escribe una función que calcule el hiperfactorial de un numero.
```

```
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Desarrollo_de_aplicaciones_para_Análisis_de_Datos\Funciones\14-hyperfactorial> python main.py
El hiperfactorial de 5 es: 3413
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Desarrollo_de_aplicaciones_para_Análisis_de_Datos\Funciones\14-hyperfactorial>
```

15. fibonacci-generator Este código genera los primeros `n` números de la secuencia de Fibonacci utilizando un generador con `yield`.

```
Desarrollo_de_aplicaciones_para_Analisis_de_Datos > Funciones >  
unmetro, 13 minutes ago | 1 author (unmetro)  
1 unmetro, 13 minutes ago • tu tia wey  
2 v def fibonacci(n):  
3     a, b = 0, 1  
4 v     for _ in range(n):  
5         yield a  
6         a, b = b, a + b  
7  
8     # Ejemplo de uso  
9     n = 10  
10    resultado = list(fibonacci(n))  
11    print(resultado)  
12  
  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Funciones\15-fibonacci-generator/main.py"  
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Funciones\15-fibonacci-generator/main.py"
```

16. palindrome-recursive Este código verifica si un texto es un palíndromo utilizando recursión, ignorando espacios y diferencias entre mayúsculas y minúsculas.

```
1 unmetro, 13 minutes ago • tu tia wey  
2 def is_palindrome(text):  
3     # Limpiamos el texto: eliminamos espacios y convertimos a minúsculas  
4     text = ''.join(text.split()).lower()  
5  
6     # Caso base: Si la longitud del texto es 0 o 1, es un palíndromo  
7     if len(text) <= 1:  
8         return True  
9  
10    # Verificamos si el primer y último carácter coinciden  
11    if text[0] == text[-1]:  
12        # Llamada recursiva con el texto sin el primer y último carácter  
13        return is_palindrome(text[1:-1])  
14    else:  
15        # Si no coinciden, no es un palíndromo  
16        return False  
17  
18    # Ejemplos de uso  
19    print(is_palindrome("Mom")) # Debería imprimir: True  
20    print(is_palindrome("Son")) # Debería imprimir: False  
21  
  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Funciones\16-palindrome-recursive/main.py"  
True  
False  
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre\Funciones\16-palindrome-recursive/main.py"
```

17. deco-positive Este código define un decorador `@assert_positive` que valida que todos los argumentos numéricos de una función sean positivos. Si algún argumento no es positivo, devuelve `None`; de lo contrario, ejecuta la función normalmente.

```
Desarrollo_de_aplicaciones_para_Analisis_de_Datos > Funciones > 17-deco-positive > main.py > ...
1  from functools import wraps
2
3  def assert_positive(func):
4      @wraps(func)
5      def wrapper(*args, **kwargs):
6          # Verificar si todos los argumentos numéricos son positivos
7          for arg in list(args) + list(kwargs.values()):
8              if isinstance(arg, (int, float)) and arg <= 0:
9                  return None # Si algún argumento no es positivo, devolver None
10             # Si todos los argumentos numéricos son positivos, ejecutar la función
11             return func(*args, **kwargs)
12     return wrapper
13
14 # Ejemplo de uso
15 @assert_positive
16 def add(x, y):
17     return x + y
18
19 print(add(2, 3)) # Debería imprimir: 5
20 print(add(-2, 3)) # Debería imprimir: None
21
```

False  
PS C:\Users\HP\Documents\4t...  
nciones/17-deco-positive/ma...  
5  
None  
PS C:\Users\HP\Documents\4t...

18. slice-recursive Este código define la función `rslice` que divide recursivamente una cadena en trozos del tamaño especificado. Si la cadena está vacía, devuelve una lista vacía. De lo contrario, toma el primer segmento y llama recursivamente a la función con el resto del texto.

```
1 unmetro, 13 minutos ago • tu tía wey
2 def rslice(text, size):
3     # Caso base: Si la cadena está vacía, devolver una lista vacía
4     if not text:
5         return []
6
7     # Dividir el texto en el primer trozo y el resto
8     return [text[:size]] + rslice(text[size:], size)
9
10 # Ejemplo de uso
11 texto = 'Ahora es mejor que nunca'
12 tamaño = 4
13 resultado = rslice(texto, tamaño)
14 print(resultado)
15
None
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencias\18-slice-recursive\main.py"
['Ahor', 'a es', ' mej', 'or q', 'ue n', 'unca']
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencias\18-slice-recursive\main.py"
```

## OOP

1. Dna Este código define una clase `DNA` que representa secuencias de ADN, permitiendo varias operaciones: suma y multiplicación de secuencias, cálculo de estadísticas de bases (A, C, G, T), modificación de bases en posiciones específicas y gestión de archivos. Además, proporciona funcionalidades para combinar secuencias, identificar coincidencias y guardar/cargar secuencias desde archivos.

```
unmetro, 13 minutes ago | 1 author (unmetro)
1 class DNA:
2     def __init__(self, sequence: str):
3         """
4         Constructor que inicializa la secuencia de ADN.
5         """
6         self.sequence = sequence
7
8     def __len__(self) -> int:
9         """
10        Devuelve la longitud de la secuencia.
11        """
12        return len(self.sequence)
13
14    def __str__(self) -> str:
15        """
16        Devuelve la secuencia como cadena de texto.
17        """
18        return self.sequence
19
20    @property
21    def adenines(self) -> int:
22        """
23        Cuenta el número de adeninas (A) en la secuencia.
24        """
25        return self.sequence.count('A')
26
27    @property
28    def cytosines(self) -> int:
29        """
30        Cuenta el número de citosinas (C) en la secuencia.
31        """
32        return self.sequence.count('C')
33
34    @property
35    def guanines(self) -> int:
36        """
37        Cuenta el número de guaninas (G) en la secuencia.
38        """
39        return self.sequence.count('G')
40
```



```
@property
def thymines(self) -> int:
    """
    Cuenta el número de timinas (T) en la secuencia.
    """
    return self.sequence.count('T')

def __add__(self, other: 'DNA') -> 'DNA':
    # Determinamos la longitud de la secuencia más larga
    max_len = max(len(self.sequence), len(other.sequence))

    # Recorremos las secuencias posición por posición
    new_sequence = ''.join(
        max((self.sequence[i:i+1], other.sequence[i:i+1])) # Sin default
        for i in range(max_len)
    )

    return DNA(new_sequence)

def stats(self) -> dict[str, float]:
    """
    Calcula el porcentaje de cada base en la secuencia.
    """
    total = len(self.sequence)
    return {
        'A': self.adenines / total * 100,
        'C': self.cytosines / total * 100,
        'G': self.guanines / total * 100,
        'T': self.thymines / total * 100
    }
```

```
    }  
  
    def __mul__(self, other: 'DNA') -> 'DNA':  
        """  
        Multiplica dos secuencias, quedándose con las bases comunes  
        (posición por posición).  
        """  
        new_sequence = ''.join(  
            s if s == o else ''  
            for s, o in zip(self.sequence, other.sequence)  
        )  
        return DNA(new_sequence)  
  
    @classmethod  
    def build_from_file(cls, path: str) -> 'DNA':  
        """  
        Crea una instancia de DNA leyendo la secuencia desde un archivo.  
        """  
        with open(path, 'r') as file:  
            sequence = file.readline().strip()  
        return cls(sequence)  
  
    def dump_to_file(self, path: str) -> None:  
        """  
        Guarda la secuencia en un archivo.  
        """  
        with open(path, 'w') as file:  
            file.write(self.sequence)  
  
    def __getitem__(self, index: int) -> str:  
        """  
        Devuelve la base que ocupa la posición indicada.  
        """  
        return self.sequence[index]
```

```
4         return self.sequence[index]
5
6     def __setitem__(self, index: int, base: str) -> None:
7         """
8         Asigna una nueva base en la posición indicada.
9         Si la base no es válida, se asigna 'A' por defecto.
10        """
11        if base not in "ACGT":
12            base = 'A' # Base por defecto: adenina
13        self.sequence = (
14            self.sequence[:index] + base + self.sequence[index + 1:]
15        )
16
17    # Ejemplo de uso
18    dna1 = DNA("AGTC")
19    dna2 = DNA("TGCA")
20
21    # Imprimir las secuencias
22    print(dna1) # AGTC
23    print(dna2) # TGCA
24
25    # Sumar dos secuencias
26    suma = dna1 + dna2
27    print(suma) # TGTC
28
29    # Multiplicar dos secuencias (bases comunes)
30    producto = dna1 * dna2
31    print(producto) # G
32
33    # Mostrar estadísticas de una secuencia
34    print(dna1.stats()) # {'A': 25.0, 'C': 25.0, 'G': 25.0, 'T': 25.0}
35
36    # Modificar una base en una posición específica
37    dna1[0] = 'T'
38    print(dna1) # TGTC
39
40    # Guardar la secuencia en un archivo
41    dna1.dump_to_file('dna_sequence.txt')
42
```

```
# Ejemplo de uso
dna1 = DNA("AGTC")
dna2 = DNA("TGCA")

# Imprimir las secuencias
print(dna1) # AGTC
print(dna2) # TGCA

# Sumar dos secuencias
suma = dna1 + dna2
print(suma) # TGTC

# Multiplicar dos secuencias (bases comunes)
producto = dna1 * dna2
print(producto) # G

# Mostrar estadísticas de una secuencia
print(dna1.stats()) # {'A': 25.0, 'C': 25.0, 'G': 25.0, 'T': 25.0}

# Modificar una base en una posición específica
dna1[0] = 'T'
print(dna1) # TGTC

# Guardar la secuencia en un archivo
dna1.dump_to_file('dna_sequence.txt')

# Cargar una secuencia desde un archivo
dna3 = DNA.build_from_file('dna_sequence.txt')
print(dna3) # TGTC
```

```
jetos/1-dna/main.py
AGTC
TGCA
TGTC
G
{'A': 25.0, 'C': 25.0, 'G': 25.0, 'T': 25.0}
TGTC
TGTC
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Cienc
```

2. Istack Este código implementa una clase `IntegerStack` que representa una pila de enteros con operaciones básicas como `*push*`, `*pop*` y `*peek*`. Además, se hace iterable mediante un iterador personalizado `IntegerStackIterator`, que permite recorrer los elementos de la pila desde el tope hacia la base usando un bucle `for`.

```

desarrollo_de_aplicaciones_para_Analisis_de_Datos > Objetos > 2-istack > main.py > IntegerStackIterator
unmetro, 14 minutos ago | 1 author (unmetro)

1 class IntegerStackIterator:
2     def __init__(self, stack: 'IntegerStack'):
3         """Inicializa el iterador con la pila dada."""
4         self._stack = stack
5         self._index = len(stack.items) - 1 # Empieza desde el top de la pila
6
7     def __next__(self) -> int:
8         """Retorna el siguiente elemento en la pila."""
9         if self._index >= 0:
10            value = self._stack.items[self._index]
11            self._index -= 1 # Mueve el índice hacia abajo en la pila
12            return value
13        else:
14            raise StopIteration # No hay más elementos para iterar
15
16
17 # Para hacer que la clase IntegerStack sea iterable, añadimos el método __iter__:
18 unmetro, 14 minutos ago | 1 author (unmetro)
19 class IntegerStack:
20     def __init__(self, *, max_size: int = 10):
21         self.items = []
22         self.max_size = max_size
23
24     def push(self, item: int) -> bool:
25         if len(self.items) < self.max_size:
26             self.items.append(item)
27             return True
28         else:
29             return False
30
31     def pop(self) -> int:
32         if self.items:
33             return self.items.pop()
34         else:
35             return None
36
37     def peek(self) -> int:
38         if self.items:
39             return self.items[-1]
40         else:
41             return None
42
43     def is_empty(self) -> bool:
44         return len(self.items) == 0
45
46     def is_full(self) -> bool:
47         return len(self.items) == self.max_size
48
49     def size(self) -> int:
50         return len(self.items)
51
52     def __iter__(self):
53         """Devuelve un iterador para la pila."""
54         return IntegerStackIterator(self)

```

```
# Ejemplo de uso:
stack = IntegerStack(max_size=5)
stack.push(3)
stack.push(7)
stack.push(1)
stack.push(6)
stack.push(2)

# Ahora podemos iterar sobre la pila con un bucle for
for item in stack:
    print(item) # Imprimirá los elementos de la pila desde el top hacia el fondo
```

```
PS C:\Users\HP\Documents\4to Sem
jetos/2-istack/main.py"
2
6
1
7
3
PS C:\Users\HP\Documents\4to Sem
```

3. Iqueue Este código define una clase `IntegerQueue` que implementa una cola de enteros con un tamaño máximo ajustable. Permite agregar (enqueue) y remover (dequeue) elementos, expandir el tamaño de la cola, guardar y cargar su contenido desde un archivo, y sumar colas. Además, proporciona un iterador personalizado para recorrer sus elementos.

```
unmetro, 15 minutos ago | 1 author (unmetro)
class IntegerQueue:
    def __init__(self, *, max_size: int = 10):
        self.items = [] # Lista para almacenar los elementos de la cola
        self.max_size = max_size # Tamaño máximo de la cola

    def enqueue(self, item: int) -> bool:
        if self.is_full():
            return False # La cola está llena, no se puede añadir más elementos
        self.items.append(item)
        return True # Se añadió con éxito

    def dequeue(self) -> int:
        if self.is_empty():
            raise IndexError("La cola está vacía.") # Error si la cola está vacía
        return self.items.pop(0) # Eliminamos y devolvemos el primer elemento

    def head(self) -> int:
        if self.is_empty():
            raise IndexError("La cola está vacía.")
        return self.items[0] # Devolvemos el primer elemento sin eliminarlo

    def is_empty(self) -> bool:
        return len(self.items) == 0 # Verdadero si la cola está vacía

    def is_full(self) -> bool:
        return len(self.items) >= self.max_size # Verdadero si la cola está llena

    def expand(self, factor: int = 2) -> None:
        self.max_size *= factor # Expandimos el tamaño máximo

    def dump_to_file(self, path: str) -> None:
        with open(path, "w") as f:
            f.write(",".join(map(str, self.items))) # Escribimos los elementos en el archivo
```

```
@classmethod
def load_from_file(cls, path: str) -> "IntegerQueue":
    with open(path, "r") as f:
        elements = list(map(int, f.read().split(",")))
    queue = cls(max_size=len(elements))
    queue.items = elements
    return queue

def __getitem__(self, index: int) -> int:
    return self.items[index] # Devuelve el elemento en la posición indicada

def __setitem__(self, index: int, item: int) -> None:
    self.items[index] = item # Asigna el valor en la posición indicada

def __add__(self, other: "IntegerQueue") -> "IntegerQueue":
    new_queue = IntegerQueue(max_size=self.max_size + other.max_size)
    new_queue.items = self.items + other.items # Concatenamos los elementos de ambas colas
    return new_queue

def __iter__(self):
    return IntegerQueueIterator(self) # Devolvemos un iterador
```

```
unmetro, 15 minutes ago | 1 author (unmetro)
class IntegerQueueIterator:
    def __init__(self, queue: IntegerQueue):
        self.queue = queue # Referencia a la cola
        self.index = 0 # Índice para el recorrido

    def __next__(self) -> int:
        if self.index >= len(self.queue.items):
            raise StopIteration # Si llegamos al final, detenemos la iteración
        item = self.queue.items[self.index]
        self.index += 1
        return item # Devolvemos el siguiente elemento
```

```
# Prueba del código
if __name__ == "__main__":
    # Crear una cola e insertar elementos
    cola = IntegerQueue(max_size=5)
    cola.enqueue(1)
    cola.enqueue(2)
    cola.enqueue(3)

    # Mostrar el primer elemento
    print("Head:", cola.head()) # Head: 1

    # Eliminar un elemento
    print("Dequeue:", cola.dequeue()) # Dequeue: 1

    # Expandir la cola
    cola.expand(2)
    print("Nuevo tamaño máximo:", cola.max_size) # Nuevo tamaño máximo: 10

    # Guardar en archivo y cargar desde archivo
    cola.dump_to_file("cola.txt")
    nueva_cola = IntegerQueue.load_from_file("cola.txt")
    print("Elementos de la nueva cola:", nueva_cola.items) # Elementos: [2, 3]

    # Sumar dos colas
    cola2 = IntegerQueue(max_size=3)
    cola2.enqueue(4)
    cola2.enqueue(5)
    cola_suma = cola + cola2
    print("Cola suma:", cola_suma.items) # Cola suma: [2, 3, 4, 5]

    # Iterar sobre la cola
    for item in cola:
        print("Iterando:", item)
```

```
3
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-d
jetos/3-iqueue.py"
Head: 1
Dequeue: 1
Nuevo tamaño máximo: 10
Elementos de la nueva cola: [2, 3]
Cola suma: [2, 3, 4, 5]
Iterando: 2
Iterando: 3
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-d
```

4. **Date** Este código define una clase `Date` que representa una fecha personalizada con varias funcionalidades adicionales. Permite operaciones como comparar fechas, sumar o restar días, calcular el día de la semana utilizando el algoritmo de Zeller, verificar si es fin de semana y obtener la fecha en diferentes formatos. Además, permite convertir fechas a un formato basado en días acumulados y viceversa. También se pueden realizar operaciones aritméticas entre dos fechas y manipularlas fácilmente.



```

You, 38 seconds ago | 1 author (You)
from datetime import date
You, 38 seconds ago * hola

You, 38 seconds ago | 1 author (You)
class Date(object):

    DIAS_SEMANA = ["DOMINGO", "LUNES", "MARTES", "MIÉRCOLES", "JUEVES", "VIERNES", "SÁBADO"]
    MESES = ["ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO", "JUNIO", "JULIO", "AGOSTO", "SEPTIEMBRE", "OCTUBRE", "NOVIEMBRE", "DICIEMBRE"]

    def __init__(self, day: int, month: int, year: int) -> None:
        if day < 1 or day > 31:
            self.day = 1
        else:
            self.day = day
        if month < 1 or month > 12:
            self.month = 1
        else:
            self.month = month
        if year < 1900 or year > 2050:
            self.year = 1900
        else:
            self.year = year

    @staticmethod
    def is_leap_year(year: int) -> bool:
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            return True
        else:
            return False

    @staticmethod
    def get_days_in_month(month: int, year: int) -> int:
        if month == 2:
            if Date.is_leap_year(year):
                return 29
            return 28
        if month in [4, 6, 9, 11]:
            return 30
        return 31

    def days_elapsed_full_years(self) -> int:
        days = 0
        for year in range(1900, self.year):
            if Date.is_leap_year(year):
                days += 366
            else:
                days += 365
        return days

    def days_elapsed_full_months(self) -> int:
        days = 0
        for month in range(1, self.month):
            days += Date.get_days_in_month(month, self.year)
        return days

    def get_delta_days(self) -> int:
        days = self.days_elapsed_full_years()
        days += self.days_elapsed_full_months()
        days += self.day - 1
        return days

```

```

# Zeller Algorithm

def weekday(self) -> int:
    month = self.month
    year = self.year
    if month < 3:
        month += 12
        year -= 1

    q = self.day
    m = month
    k = year % 100 # Últimos dos dígitos del año
    j = year // 100 # Primeros dos dígitos del año

    # Aplicando la fórmula de Zeller
    f = q + (13 * (m + 1)) // 5 + k + (k // 4) + (j // 4) - 2 * j
    week_day = f % 7

    # Ajustar el resultado al formato Domingo(0), Lunes(1), ..., Sábado(6)
    # Zeller devuelve: Sábado(0), Domingo(1), ..., Viernes(6)
    # Queremos: Domingo(0), Lunes(1), ..., Sábado(6)
    week_day = (week_day + 6) % 7 # Ajuste para que Domingo sea 0

    return week_day

def is_weekend(self) -> bool:
    return self.weekday() == 0 or self.weekday() == 6

def short_date(self) -> str:
    return f"{self.day:02}/{self.month:02}/{self.year:04}"

def __str__(self) -> str:
    week_day = self.DIAS_SEMANA[self.weekday()]
    month_name = self.MESES[self.month - 1]
    return f"{week_day} {self.day} DE {month_name} DE {self.year}"

```

```
def __add__(self, days_to_add: int) -> 'Date':
    new_day = self.day
    new_month = self.month
    new_year = self.year

    while days_to_add > 0:
        days_current_month = self.get_days_in_month(new_month, new_year)

        if new_day + days_to_add <= days_current_month:
            new_day += days_to_add
            break
        else:
            days_to_add -= (days_current_month - new_day + 1)
            new_day = 1
            new_month += 1
            if new_month > 12:
                new_month = 1
                new_year += 1
            new_day += 1
    return Date(new_day, new_month, new_year)

def to_days(self) -> int:
    total_days = 0
    for year in range(1, self.year):
        total_days += 366 if self.is_leap_year(year) else 365
    for month in range(1, self.month):
        total_days += self.get_days_in_month(month, self.year)
    total_days += self.day
    return total_days
```

```
25
26 def from_days(self, total_days: int) -> 'Date':
27     year = 1
28     while total_days > (366 if self.is_leap_year(year) else 365):
29         total_days -= 366 if self.is_leap_year(year) else 365
30         year += 1
31
32     month = 1
33     while total_days > self.get_days_in_month(month, year):
34         total_days -= self.get_days_in_month(month, year)
35         month += 1
36
37     day = total_days
38     return Date(day, month, year)
39
40 def __sub__(self, other):
41     if isinstance(other, Date):
42         return abs(self.to_days() - other.to_days())
43     elif isinstance(other, int):
44         total_days = self.to_days() - other
45         return self.from_days(total_days)
46
47 def __eq__(self, other: object) -> bool:
48     if not isinstance(other, Date):
49         return False
50     return self.day == other.day and self.month == other.month and self.year == other.year
51
52 def __gt__(self, other: object) -> bool:
53     if not isinstance(other, Date):
54         return False
55     if self.year > other.year:
56         return True
57     elif self.year == other.year:
58         if self.month > other.month:
59             return True
60         elif self.month == other.month:
61             return self.day > other.day
62     return False
```

```
163
164     def __lt__(self, other: object) -> bool:
165         if not isinstance(other, Date):
166             return False
167         if self.year < other.year:
168             return True
169         elif self.year == other.year:
170             if self.month < other.month:
171                 return True
172             elif self.month == other.month:
173                 return self.day < other.day
174             return False
175
176 if __name__ == "__main__":
177     date1 = Date(day=1, month=3, year=1979)
178     date2 = Date(day=24, month=6, year=1984)
179     print(date1)
180     print(date2)
181     print(date1 == date2)
182     print(date1 < date2)
183     print(date1 > date2)
184     print(date1 + 10)
185     print(date1 - 10)
186     print(date1 - date2)
187     print(date1.is_weekend())
188     print(date2.is_weekend())
189     print(date1.short_date())
190     print(date2.short_date())
191     print(date1.weekday())
192     print(date2.weekday())
193     print(date1.to_days())
194     print(date2.to_days())
195     print(date1.from_days(28913))
196     print(date2.from_days(4748))
197
198
```

```
Iterando: 2
Iterando: 3
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Dat
jetos/4-date/main.py"
JUEVES 1 DE MARZO DE 1979
DOMINGO 24 DE JUNIO DE 1984
False
True
False
DOMINGO 11 DE MARZO DE 1979
LUNES 19 DE FEBRERO DE 1979
1942
False
True
01/03/1979
24/06/1984
4
0
722509
724451
MIÉRCOLES 28 DE FEBRERO DE 1900
LUNES 31 DE DICIEMBRE DE 1900
PS C:\Users\HP\Documents\4to Semestre\4to-Semestre-Ciencia-de-Dat
```

## EXCEPTIONS

1. **Poker-Card** Este código define una clase `Card` que representa cartas de poker, permitiendo validarlas, compararlas y sumarlas. También incluye una excepción personalizada `InvalidCardError` para manejar entradas inválidas. Provee métodos para crear cartas, obtener los palos disponibles, y generar todas las cartas de un palo específico. Finalmente, permite sumar dos cartas, ajustando su valor si supera 13.

```

desarrollo_de_aplicaciones_para_Analisis_de_Datos > Excepciones > cartas.py > ...
1 class InvalidCardError(Exception):
2     """Excepción personalizada para cartas inválidas."""
3
4     def __init__(self, message: str = ''):
5         super().__init__(f'Invalid card: {message if message else "Invalid card"}')
6
7     def __str__(self):
8         return self.args[0]
9
10 class Card:
11     """Clase que representa una carta de poker."""
12
13     # Definimos los valores posibles para las cartas
14     VALUE_MAP = {
15         'A': 1, '2': 2, '3': 3, '4': 4, '5': 5,
16         '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
17         'J': 11, 'Q': 12, 'K': 13
18     }
19
20     SUITS = ['♠', '♦', '♥', '♣']
21
22     def __init__(self, value: int | str, suit: str):
23         # Validamos el valor
24         if isinstance(value, str) and value in self.VALUE_MAP:
25             self.value = self.VALUE_MAP[value]
26         elif isinstance(value, int) and 1 <= value <= 13:
27             self.value = value
28         else:
29             raise InvalidCardError(f'{repr(value)} is not a supported value')
30
31         # Validamos el palo
32         if suit not in self.SUITS:
33             raise InvalidCardError(f'{repr(suit)} is not a supported suit')
34
35         self.suit = suit
36
37     def __repr__(self):
38         """Devuelve el glifo de la carta."""
39         return f'{self.value} {self.suit}'
40
41     def __eq__(self, other: object) -> bool:
42         """Indica si dos cartas son iguales."""
43         if not isinstance(other, Card):
44             return False
45         return self.value == other.value and self.suit == other.suit
46
47     def __lt__(self, other: 'Card') -> bool:
48         """Indica si una carta es menor que otra."""
49         return self.value < other.value
50
51     def __gt__(self, other: 'Card') -> bool:
52         """Indica si una carta es mayor que otra."""
53         return self.value > other.value

```

```

def __add__(self, other: 'Card') -> 'Card':
    """Devuelve una nueva carta como suma de dos cartas."""
    new_value = self.value + other.value
    new_value = 1 if new_value > 13 else new_value # Se convierte en As si supera 13

    new_suit = self.suit if self > other else other.suit
    return Card(new_value, new_suit)

@classmethod
def get_available_suits(cls) -> str:
    """Devuelve los palos disponibles."""
    return ', '.join(cls.SUITS)

@classmethod
def get_cards_by_suit(cls, suit: str):
    """Genera todas las cartas de un palo específico."""
    if suit not in cls.SUITS:
        raise InvalidCardError(f'{repr(suit)} is not a supported suit')
    for value in cls.VALUE_MAP.keys():
        yield Card(value, suit)

# Ejemplo de uso
try:
    carta1 = Card('A', '♠')
    carta2 = Card(13, '♦')
    print(carta1) # A ♠
    print(carta2) # 13 ♦
    print(carta1 + carta2) # 1 ♠ (porque A + K es 14, y se convierte en As)
except InvalidCardError as e:
    print(e)

```

```

PS C:\Users\H...
ceptiones/cart
1 ♠
13 ♦
1 ♦
PS C:\Users\H...

```

## CONCLUSIONES

A lo largo de esta práctica se trabajó con diversos conceptos fundamentales de programación, incluyendo estructuras de control, funciones recursivas, generadores, y programación orientada a objetos. Cada ejercicio permitió explorar un aspecto particular, desde manipulación de listas y validación de datos hasta el uso de decoradores y excepciones personalizadas. Además, se implementaron algoritmos clave como el ordenamiento por burbuja, verificación de palíndromos y cálculo de números de Fibonacci, lo que reforzó la comprensión de técnicas de resolución de problemas. Esta práctica fue fundamental para aplicar conocimientos teóricos de forma práctica, mejorando la capacidad de diseñar soluciones eficientes y estructuradas en Python.