SPRINT 7: NOCIONES Y CONOCIMIENTOS BÁSICOS EN PYTHON. ESTRUCTURAS DE DATOS Y DE CONTROL

NIVEL 1

Ejercicio 1. Calculadora del índice de masa corporal. Escribe una función que calcule el IMC ingresado por el usuario/a, es decir, quien lo ejecute deberá ingresar estos datos. La función debe clasificar el resultado en sus respectivas categorías.

Explicación del código:

- defino la función de cálculo de IMC y de clasificación en categorías según el resultado; incluyo print del resultado, redondeando IMC a 1 decimal (:.1f)

```
def IMC_categoria(peso,altura):
    IMC = peso/(altura**2)

    if IMC < 18.5:
        categoria = 'bajo peso'
    elif IMC < 25:
        categoria = 'peso normal'
    elif IMC < 30:
        categoria = 'sobrepeso'
    else:
        categoria = 'obesidad'

    print(f'Su índice de masa corporal es {IMC:.1f} y pertenece a la categoría de {categoria}.')
```

- input para que el usuario introduzca su peso en kg
- input para que el usuario introduzca su altura en metros
- añado los 4 puntos de control para validar los datos del input (explicado en el apartado EXTRA)
- print de los datos introducidos por el usuario
- llamo a la función, con los argumentos de peso y altura del input

```
peso = input('Primero, introduzca su peso en kg (ej. 60.5): ')
altura = input('Ahora, introduzca su altura en m (ej. 1.80): ')

peso = float(peso)
altura = float(altura)

if peso <= 0 or peso > 550:
    raise ValueError
if altura <=0 or altura > 3:
    raise ValueError

print(f'Datos introducidos: {peso} kg y {altura} m.')

IMC_categoria(peso, altura)
```

EXTRA: validación de los valores introducidos en el input

- tomo en consideración que el usuario puede introducir peso o altura en valor negativo/cero o unos valores físicamente poco probables (ej: superiores a los máximos registrados en el record guiness)
    - en esos casos, saltará error ValueError (raise ValueError)
- uso **try-except** para printar mensaje de error en caso de que el usuario introduzca los datos en un formato incorrecto (ej: coma en lugar de punto para decimales; str en lugar de int; peso o altura <= 0 o muy elevados)

- **try**: cuerpo del código; se ejecutará completo siempre y cuando el usuario introduzca valores válidos; el código se interrumpirá en caso de no superar los siguientes puntos:
  - 1r punto: casting peso a float
  - 2o punto: casting altura a float
  - 3r punto: peso dentro del rango lógico
  - 4o punto: altura dentro del rango lógico
- **except ValueError** (error que salta cuando no se puede convertir input a float o cuando se está fuera del rango permitido para el peso/altura): print mensaje de error

```
except ValueError:
    print(f'''Ha introducido {peso} kg y {altura} m.
Estos datos no cumplen con el formato permitido. Tenga en cuenta que:
    · el peso y la altura deben ser valores numéricos
    · los decimales deben escribirse con . (ej. 60.5 kg)
    · el rango de peso aceptado es: entre 0 y 550 kg
    · el rango de altura aceptada es: entre 0 y 3 m
Por favor, ejecuta de nuevo el programa para realizar el cálculo de IMC.''')
```

CORRECCIONES POR REALIZAR:

- Implementar un bucle que permita a los usuarios realizar múltiples cálculos de IMC en una sola ejecución del programa, en lugar de tener que reiniciar el programa cada vez.
- Separar la lógica de cálculo del IMC de la lógica de interacción con el usuario (por ejemplo, tener una función puramente para calcular el IMC, y otra función para pedir datos al usuario y mostrar los resultados)

<u>Ejercicio 2.</u> Conversor de temperaturas. Existen diversas unidades de temperatura utilizadas en distintos contextos y regiones. Las más comunes son Celsius (°C), Fahrenheit (°F) y Kelvin (K). También existen otras unidades como Rankine (°Ra) y Réaumur (°Re). Selecciona al menos 2 y construye el conversor.

Para mi programa, seleccioné 3 unidades de temperatura: Celsius (°C), Fahrenheit (°F) y Kelvin (K). El usuario decide en qué unidad introducirá la temperatura y obtendrá como resultado el equivalente en las otras dos unidades.

Explicación del código

- defino la función de conversión de temperatura de Celsius a Fahrenheit, Celsius a Kelvin, etc.

```python
def celsius_to_kelvin(temp):
    return temp + 273.15
def celsius_to_fahrenheit(temp):
    return temp * 9/5 + 32
def fahrenheit_to_celsius(temp):
    return temp - 32 * 5/9
def fahrenheit_to_kelvin(temp):
    return fahrenheit_to_celsius(temp) + 273.15
def kelvin_to_celsius(temp):
    return temp - 273.15
def kelvin_to_fahrenheit(temp):
    return kelvin_to_celsius(temp) * 9/5 + 32
```

- input para que el usuario introduzca la unidad de la temperatura que desea convertir (C, F o K)
- si la unidad introducida es C o F, añado símbolo ° para mejorar el formato de los prints
- input para que el usuario introduzca el valor de la temperatura que desea convertir
- añado los 3 puntos de control para validar los datos del input (explicado en el apartado EXTRA)
- print de los datos introducidos por el usuario

```python
unidad = input('¿Qué unidad de temperatura desea convertir? Teclee la letra C para Celsius, F para Fahrenheit o K para Kelvin.').capitalize()
if unidad == 'C' or unidad == 'F':
    unidad = '°' + unidad

valor = input('Ahora, indique el valor de temperatura que desea convertir, utilizando . como separador de decimales (ej. 20.5): ')

if unidad not in {'°C', '°F', 'K'}:
    raise ValueError

valor = float(valor)

if unidad == '°C' and valor < -273.15:
    raise ValueError
elif unidad == '°F' and valor < -459.67:
    raise ValueError
elif unidad == 'K' and valor < 0:
    raise ValueError

print(f'Temperatura introducida: {valor} {unidad}')
```

- print del resultado en función de la unidad introducida (if - elif - else), llamo a las funciones definidas al inicio del programa

- resultado de la conversión redondeado a 2 decimales (:.2f)

```
if unidad == '°C':
    print(f'{valor} Celsius ({unidad}) equivale a {celsius_to_kelvin(valor):.2f} Kelvin (K).')
    print(f'{valor} Celsius ({unidad}) equivale a {celsius_to_fahrenheit(valor):.2f} Fahrenheit (°F).')
elif unidad == '°F':
    print(f'{valor} Fahrenheit ({unidad}) equivale a {fahrenheit_to_celsius(valor):.2f} Celsius (°C).')
    print(f'{valor} Fahrenheit ({unidad}) equivale a {fahrenheit_to_kelvin(valor):.2f} Kelvin (K).')
else:
    print(f'{valor} Kelvin ({unidad}) equivale a {kelvin_to_celsius(valor):.2f} Celsius (°C).')
    print(f'{valor} Kelvin ({unidad}) equivale a {kelvin_to_fahrenheit(valor):.2f} Fahrenheit (°F).')
```

EXTRA: validación de los valores introducidos en el input**

- tomo en consideración que el usuario puede introducir unidad de temperatura en formato incorrecto (si escribe toda la unidad completa en lugar de unicamente la letra inicial) o una para la que no tenemos función de conversión definida (otras unidades de temperatura); también tomo en consideración que el valor de temperatura no puede ser inferior al cero absoluto
    - en esos casos, saltará error ValueError (raise ValueError)
- uso **try-except** para printar mensaje de error en caso de que el usuario introduzca los datos en un formato incorrecto (ej: unidad incorrecta; coma en lugar de punto para decimales; str en lugar de int; < cero absoluto)
    - **try**: cuerpo del código; se ejecutará completo siempre y cuando el usuario introduzca valores válidos; el código se interrumpirá en caso de no superar los siguientes puntos (definidos después de los inputs en el bloque try):
        - 1r punto: unidad diferente a '°C', '°F' o 'K'
        - 2o punto: casting valor temperatura a float
        - 3r punto: valor inferior al cero absoluto
    - **except ValueError**: print mensaje de error

```
except ValueError:
    print(f'''Ha introducido: {valor} {unidad}.
Estos datos no cumplen con el formato permitido. Tenga en cuenta que:
    · las unidades de conversión son: Celsius (°C), Fahrenheit (°F) o Kelvin (K)
    · el valor de la temperatura debe ser numérico
    · los decimales deben escribirse con . (ej. 20.5 °C)
    · el valor de temperatura no puede ser inferior al cero absoluto: -273.15 °C, -459.67 °F o 0 K
Por favor, ejecuta de nuevo el programa para realizar la conversión de temperatura.''')
```

CORRECCIONES POR REALIZAR: encapsular la lógica de decisión sobre qué conversión realizar en una función separada, es decir, definir otra función que realice la conversión entre las temperaturas.

<u>Ejercicio 3</u>. Contador de palabras de un texto. Escribe una función que dado un texto, muestre las veces que aparece cada palabra.

Explicación del código

- defino la función de limpiador de texto introducido, que contiene dos partes fundamentales:
  - limpieza del texto introducido: para quitar los signos de puntuación y posibles números
    - defino los posibles signos de puntuación que se pueden encontrar en un texto (entre ''' para poder incluir ' y " en la cadena)
    - defino números
    - creo bucle for que recorre el texto introducido; en caso de encontrarse con un signo de puntuación o con un número, lo reemplaza por un espacio en blanco
    - decido reemplazarlo por espacio en blanco (' ') en lugar de simplemente eliminarlo ('') para separar correctamente las palabras
      - ej. "texto_separado" → queda como "texto separado" (dos palabras separadas) y no "textoseparado" (una palabra)
    - una vez el texto esta limpio, convierto en lower todas las palabras para evitar que la misma palabra se cuente como diferente por estar en mayúscula
      - ej. "uno Uno UNO" → queda como "uno uno uno" (uno: 3 veces) y no "uno Uno UNO" (uno: 1 vez; Uno: 1 vez; UNO: 1 vez)
  - separación del texto en una lista de palabras (elijo lista porque se permiten duplicados y se puede modificar)
    - separo el texto con el método split → se crea una lista con las palabras

```
def limpiador_texto(texto_introducido):
    print(f'Ha introducido el siguiente texto: {texto_introducido}')

    #limpieza del texto introducido
    signos = '''¡!¿?([{)]}.,;:'"<>+*=/@#€$§%^&-_~'''
    numeros = '0123456789'

    for x in texto_introducido:
        if (x in signos) or (x in numeros):
            texto_introducido = texto_introducido.replace(x, ' ')

    texto_limpio = texto_introducido.lower()

    #separación del texto en lista de palabras
    lista_palabras = texto_limpio.split()

    print(f'Su texto contiene {len(lista_palabras)} palabras.')
    return lista_palabras
```

- defino la función del contador de palabras:
  - creo un set vacío
  - creo bucle for que recorrerá la lista con las palabras, hará el recuento de la palabra y las añadirá al set; si la palabra ya está dentro del set, el bucle vuelve al inicio sin printar resultado (print(f'{palabra}: {recuento}'))

- la funcionalidad del set, por tanto, es almacenar las palabras únicas del texto y evitar printar la misma palabra dos o más veces en el resultado del bucle

```python
def recuento_palabras(lista_palabras):
    print('A continuación, se concreta la cantidad de veces que se aparece cada palabra:')

    palabras_unicas = set()
    for palabra in lista_palabras:
        recuento = lista_palabras.count(palabra)
        if palabra in palabras_unicas:
            continue
        else:
            palabras_unicas.add(palabra)
        print(f'{palabra}: {recuento}')
```

- input para que el usuario introduzca un texto
- creo una variable llamada palabras_texto y llamo a la función de limpiador de texto → palabras_texto almacena las palabras del texto introducido en el input
- llamo a la función de contador de palabras de esa lista

Ejemplo del texto introducido: "Estudios: clave del éxito. ¡Investiga! ¡Analiza! ¡Aprende! ¡Hoy, el 80% de los estudiantes acceden a recursos digitales para mejorar sus estudios!"

CORRECCIONES POR REALIZAR:

- Emplear una expresión regular con re.sub para eliminar signos de puntuación y números. Esto puede reducir la cantidad de operaciones necesarias al limpiar el texto, ya que reemplazarás todos los caracteres no deseados en una sola pasada.
- En lugar de usar list.count dentro de un bucle, que es computacionalmente costoso, utilizar un diccionario para contar las palabras mientras itero sobre ellas una sola vez. Esto es mucho más eficiente, especialmente para textos más largos.

<u>Ejercicio 4</u>. Diccionario inverso. Resulta que el cliente tiene una encuesta muy antigua que se almacena en un diccionario y los resultados los necesita a la inversa, es decir, intercambiados las claves y valores. Los valores y claves en el diccionario original son únicos; si éste no es el caso, la función debería imprimir un mensaje de advertencia.

NOTA: por definición, en un diccionario las claves no pueden tener duplicados, es decir, son únicas; los valores, en cambio, sí que pueden repetirse.

Explicación del código

- definición de la función diccionario invertido:
  - creo un diccionario vacío
  - creo bucle for que recorrerá las claves y los valores del diccionario original
    - dado que en Python se distingue entre mayúsculas y minúsculas, transformo la primera letra de las claves y los valores en mayúsculas
      - para evitar duplicados del estilo: 'valor3' y 'Valor3'
  - añado condicional para que salte mensaje de valor duplicado; si el valor ya está como clave en el diccionario inverso, el bucle empieza desde el inicio (continue) y no se actualiza clave:valor del diccionario inverso; por tanto, en el diccionario inverso aparecerán la primera combinación valor:clave única encontrada en el diccionario original
    - diccionario original: {'clave3': 'valor3', 'clave4': 'valor3'}
    - diccionario inverso: {'valor3': 'clave3'} en lugar de {'valor3': 'clave4'}
      - el valor para la clave 'valor3' se actualizaría si no se añade *continue* en el condicional
  - añado la clave y el valor del diccionario original al diccionario inverso, intercambiados
  - print del diccionario inverso

```
def diccionario_invertido(diccionario_original):
    diccionario_inverso = {}
    for clave, valor in diccionario_original.items():
        clave = clave.title()
        valor = valor.title()
        if valor in diccionario_inverso.keys():
            print(f'El diccionario original contiene el valor "{valor}" duplicado. No se ha incluido la combinación "{clave}: {valor}" al diccionario inverso.')
            continue
        else:
            diccionario_inverso[valor] = clave
    print(f'El diccionario inverso es: {diccionario_inverso}')
```

- creo una variable para el diccionario original
- llamo a la función diccionario invertido

CORRECCIONES POR REALIZAR: añadir una función o mecanismo para reportar todos los duplicados al final de la ejecución en lugar de en cada iteración. Esto podría hacerse almacenando los duplicados en una lista y mostrándolas todos juntos al final.

NIVEL 2

<u>Ejercicio 1</u>. Diccionario inverso con duplicados. Continuando con el ejercicio 4 del nivel 1: al cliente se olvidó comentar un detalle y resulta que los valores en el diccionario original pueden duplicarse y más, por lo que las claves intercambiadas pueden tener duplicados. En tales casos, los valores del diccionario resultante tendrán que almacenarse como una lista. Ten en cuenta que un valor único no debe ser una lista.

NOTA: por definición, en un diccionario las claves no pueden tener duplicados, es decir, son únicas; los valores, en cambio, sí que pueden repetirse.

En caso de que un valor esté duplicado en el diccionario original, se almacenará como clave en el diccionario inverso y las claves del diccionario original se almacenarán como valores en una lista.

Explicación del código

Definición de la función diccionario invertido con duplicados:

- creo un diccionario vacío
- creo bucle for que recorrerá las claves y los valores del diccionario original
    - dado que en Python se distingue entre mayúsculas y minúsculas, transformo la primera letra de las claves y los valores en mayúsculas
        - para evitar duplicados del estilo: 'valor3' y 'Valor3'
    - defino condicional if por si el valor del diccionario original está repitido y ya se almacenó en el diccionario inverso como clave:
        - si el valor no está repetido, su clave se almacena en una lista
            - diccionario_inverso[valor] = [clave].
        - si el valor está repitido, se añade la nueva clave a la lista y se actualiza el diccionario inverso con la lista ampliada
            - i = diccionario_inverso[valor] → i.append(clave) → diccionario_inverso[valor] = i
        - de esta manera, aquellos valores que no estaban repetidos tendrán un elemento en su lista, mientras que los repetidos tendrán varios
    - como resultado, obtenemos un diccionario inverso donde todos los valores (claves del diccionario original) se almacenan en una lista
- creo bucle for que recorrerá las claves y los valores del diccionario inverso
    - si la longitud de los valores es 1, significa que unicamente hay un elemento en la lista; por tanto, actualizo ese valor para que aparezca como str y no lista
    - si la longitud es superior, significa que hay más de un elemento en la lista de valores; en ese caso no hago ningún cambio
- print del diccionario inverso

```python
def diccionario_invertido(diccionario_original):
    diccionario_inverso = {}
    for clave, valor in diccionario_original.items():
        clave = clave.title()
        valor = valor.title()
        if valor in diccionario_inverso.keys():
            i = diccionario_inverso[valor]
            i.append(clave)
            diccionario_inverso[valor] = i
        else:
            diccionario_inverso[valor] = [clave]

    for clave, valor in diccionario_inverso.items():
        i = diccionario_inverso[clave]
        if len(i) == 1:
            diccionario_inverso[clave] = i[0]
        else:
            diccionario_inverso

    print(f'El diccionario inverso es: {diccionario_inverso}')
```

Creo una variable para el diccionario original

Llamo a la función diccionario invertido

<u>Ejercicio 2</u>. Conversión de tipos de datos. El cliente recibe una lista de datos y necesita generar dos listas, la primera donde estarán todos los elementos que pudieron convertirse en flotantes y la otra donde están los elementos que no pudieron convertirse.

Ejemplo de la lista que recibe el cliente: ['1.3', 'one', '1e10', 'seven', '3-1/2', ('2',1,1.4,'not-a-number'), [1,2,'3','3.4']]

Explicación del código

- defino una variable para la lista con datos (asumo que el cliente introducirá la lista en formato correcto)
- print de la lista introducida
- creo una lista vacía para los elementos que se pueden convertir a flotantes y otra para los que no
- defino **bucle for** para recorrer los diferentes elementos de la lista:
    - uso try - except para manejar varios errores:
        - TypeError: error cuando se quiere convertir a float un elemento de tipo tupla o lista (tupla o lista dentro de la lista introducida)
        - ValueError: error cuando no se puede convertir a float a pesar de tener el elemento en int o str (ej. 'one')
    - **try**: para cada elemento interado, si se puede convertir a float, lo añado a la lista elementos_convertidos
    - **except TypeError**: defino otro bucle para que recorra elemento por elemento la lista o tupla que no se pudo convertir a float; en este caso hay dos situaciones:
        - el elemento de la lista se convierte a float → lo añado a la lista elementos_convertidos
        - el elemento de la lista no se puede convertir → genera error ValueError → utilizo otro try - except para este bucle
    - **except ValueError**: añado los elementos que no se pudieron convertir a float a la lista elementos_no_convertidos

```
elementos_convertidos = []
elementos_no_convertidos = []

for elemento in lista_cliente:
    try:
        elemento = float(elemento)
        elementos_convertidos.append(elemento)
    except TypeError:
        try:
            for x in elemento:
                x = float(x)
                elementos_convertidos.append(x)
        except ValueError:
            elementos_no_convertidos.append(x)
    except ValueError:
        elementos_no_convertidos.append(elemento)
```

- print de las dos listas (elementos_convertidos y elementos_no_convertidos)

NOTA: el elemento '3-1/2' no se convirtió a float porque está definido como str y no se puede hacer la operación de resta; si estuviese definido como 3-1/2, se realizaría la operación y se incluiría el resultado en la lista de elementos_convertidos.

CORRECCIONES POR REALIZAR: en lugar de tratar de convertir y manejar errores dentro del mismo bucle, se podría crear una función auxiliar que maneje la conversión y clasificación, y llamar a esta función recursivamente para elementos anidados.

NIVEL 3

<u>Ejercicio 1</u>. Contador y ordenador de palabras de un texto. El cliente quedó contento con el contador de palabras, pero ahora quiere leer archivos TXT y que calcule la frecuencia de cada palabra ordenadas dentro de las entradas habituales del diccionario según la letra con la que comienzan, es decir, las claves deben ir de la A a la Z y dentro de la A debemos ir de la A a la Z.

Para realizar este ejercicio, hice cambios oportunos en mi código de contador de palabras (ejercicio 1.3)

Explicación del código

- defino nueva función para abrir los archivos de texto desde la ruta del ordenador:
  - utilizo *with open() as* para que el programa haga la lectura del archivo especificado y lo cierre al finalizar el bucle for
    - encoding utf-8 para detectar correctamente los acentos
  - defino variable txt_str
  - defino bucle for para que recorra linea por linea del archivo (tal y como está en el documento original) y las almacene en la variable txt_str
  - el resultado de la función será la variable txt_str

```
def texto_archivo(ruta):
    with open(ruta, 'r', encoding='utf-8') as txt_original:
        txt_str = ''
        for linea in txt_original:
            txt_str += linea
    return txt_str
```

- modificaciones en la función limpiador_texto:
  - al haber saltos de línea en el archivo, los reemplazo por espacios en blanco (al igual que los signos de puntuación y/o números)
  - el resto del código queda igual que en el ejercicio 1.3
  - el resultado de la función es la lista de palabras del documento

```
def limpiador_texto(texto_introducido):
    #limpieza del texto introducido
    signos = '''¡!¿?([{)]}.,;:'"<>+*=/@#€$§%^&-_~'''
    numeros = '0123456789'

    for x in texto_introducido:
        if (x in signos) or (x in numeros) or (x == '\n'):
            texto_introducido = texto_introducido.replace(x, ' ')

    texto_limpio = texto_introducido.lower()

    #separación del texto en lista de palabras
    lista_palabras = texto_limpio.split()

    print(f'Su texto contiene {len(lista_palabras)} palabras.')
    return lista_palabras
```

- modificaciones en la función contador_palabras (la he modificado completamente porque la finalidad del ejercicio es almacenar el recuento de palabras en un diccionario):
    - creo un diccionario vacío llamado abcd
    - defino bucle for que recorrerá las palabras de la lista de palabras:
        - almaceno la primera letra de cada palabra (clave = palabra[0]) como *clave* en el diccionario, dejando los *valores* como diccionario vacío → abcd.update({clave:{}})
            - si la letra ya está en el diccionario, el bucle salta al siguiente condicional sin pasar por la parte del update (no uso elif porque quiero que tome dos caminos independientes y se actualice, si hace falta, tanto la clave del diccionario abcd como los valores para esa clave)
        - el diccionario abcd contendrá las letras por las cuales empiezan las palabras del documento, de modo que si hay alguna letra que no aparece en el documento (por ejemplo: ç), no aparecerá como clave
    - almaceno la palabra y su recuento (recuento = lista_palabras.count(palabra)) como valores de la clave → abcd[clave][palabra] = recuento
        - si la palabra ya forma parte de los valores, el bucle comienza una nueva iteración con otra palabra
    - el resultado de la función es el diccionario abcd que contiene como clave letras y como valores las palabras y las veces que aparece cada una en el archivo txt; sin embargo, el diccionario no está ordenado

```
def contador_palabras(lista_palabras):
    abcd = {}
    for palabra in lista_palabras:
        clave = palabra[0]
        if clave not in abcd.keys():
            abcd.update({clave:{}})
        if palabra not in abcd.values():
            recuento = lista_palabras.count(palabra)
            abcd[clave][palabra] = recuento
    return abcd
```

- defino nueva función para ordenar el diccionario:
    - creo un diccionario (diccionario_ordenado) para almacenar los datos ordenados
    - defino bucle for para recorrer las entradas del diccionario
        - uso sorted(diccionario) para ordenar las claves externas, es decir, las letras (a, b, c, d...)
        - en el diccionario_ordenado las claves aparecen alfabéticamente y los valores se almacenarán en un diccionario interno (de momento vacío)
    - defino otro bucle for interno para ordenar las claves del diccionario interno (el bucle se ejecutará para cada letra y se ordenaran las palabras alfabéticamente)
        - uso sorted(diccionario[clave]) para ordenar las claves internas, es decir, las palabras (a, agua, al, alba...)
        - asigno a cada clave interna del diccionario_ordenado el valor de recuento correspondiente (lo consigo con:

diccionario[clave][clave_interna]) → diccionario_ordenado[clave][clave_interna] = diccionario[clave][clave_interna]

NOTA: he visto que hay otras maneras de ordenar el diccionario usando las comprehensions, pero el metodo descrito con bucles fue el más intuitivo para mí.

```
def ordenador_diccionario(diccionario):
    diccionario_ordenado = {}
    for clave in sorted(diccionario):
        diccionario_ordenado[clave] = {}
        for clave_interna in sorted(diccionario[clave]):
            diccionario_ordenado[clave][clave_interna] = diccionario[clave][clave_interna]
    print(json.dumps(diccionario_ordenado, indent=2, ensure_ascii=False))
```

- una vez definidas las funciones, creo la variable ruta para insertar la ruta al documento .txt
- llamo a todas las funciones descritas