

SPRINT 9 - PYTHON

El repositorio mencionado se corresponde con los ejercicios del Sprint 9 de la especialización de Data Analyst del curso IT ACADEMY, Cibernarium (Barcelona Activa).

NIVEL 1

1.1 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. Consejo: Intenta validar los datos previamente, para que envíe un mensaje de advertencia si los datos introducidos por el usuario/a no están en el formato adecuado o no toma valores razonables.

```
1 def pedir_medida(mensaje_medida, minimo=None, maximo=None):
2     """
3         Solicita a user un valor numérico y repite bucle hasta validar que float ok y
4         que esté dentro del rango de valor minimo y/o máximo, si se ha indicado.
5
6         Argumentos:
7             - mensaje (str) que se muestra al usuario al solicitar el valor
8             - minimo (float o none)
9             - maximo (float o none)
10            Return: float con valor
11        """
12    while True:
13        try:
14            valor = float(input(mensaje_medida))
15            if minimo is not None and valor < minimo:
16                print(f"Error: el valor debe ser mayor o igual que {minimo}.")
17                continue
18            if maximo is not None and valor > maximo:
19                print(f"Error: el valor debe ser menor o igual que {maximo}.")
20                continue
21            return valor
22
23        except ValueError:
24            print("Error: escribe un valor numérico (y un punto, si deseas decimales). Vuelve a probar.")
25
26 def calculo_imc(weight_user, height_user):
27     """cálculo IMC=peso/(altura**2) con return"""
28     return weight_user / (height_user**2)
29
30 def mensaje_según_imc(imc_calculado):
31     """clasificación según IMC calculado"""
32     if imc_calculado > 30:
33         mensaje = "Tienes obesidad, contacta a tu médico."
34     elif imc_calculado >= 25:
35         mensaje = "Tienes sobrepeso, consulta con tu médico."
36     elif imc_calculado >= 18.5:
37         mensaje = "Tienes un peso normal. Enhorabuena."
38     else:
39         mensaje = "Tienes un peso demasiado bajo. Contacta a tu médico."
40     return mensaje
41
42 def main_imc():
43     height_user = pedir_medida("Introduce tu altura en metros. Por ejemplo '1.70'.", 0, 3)
44     weight_user = pedir_medida("Introduce tu peso en kg. Por ejemplo: '66.5'.", 0, 300)
45     imc_calculado = calculo_imc(weight_user, height_user)
46     mensaje = mensaje_según_imc(imc_calculado)
47     print(f'Tu IMC (índice de masa corporal) es {imc_calculado: .2f}. {mensaje}')
48
49 main_imc()
```

✓ 52.0s Python

```
Error: escribe un valor numérico (y un punto, si deseas decimales). Vuelve a probar.
Error: el valor debe ser menor o igual que 3.
Error: escribe un valor numérico (y un punto, si deseas decimales). Vuelve a probar.
Error: el valor debe ser mayor o igual que 0.
Tu IMC (índice de masa corporal) es 24.38. Tienes un peso normal. Enhorabuena.
```

Para este programa creamos 3 funciones auxiliares + 1 principal:

- "pedir_medida(mensaje_medida, minimo, maximo)"
- "calculo_imc(weight_user, height_user)"
- "mensaje_segun_imc(imc_calculado)"
- "main_imc()"

Función 1: "pedir_medida(mensaje_medida, minimo=None, maximo=None)"

- Sigue la función para solicitar un dato numérico al usuario y validar que el dato entrado sea correcto.
- Recibe un parámetro: el mensaje específico que verá el usuario cuando se le pida un dato concreto (es externo porque puede variar según el dato que se necesite solicitar).
- Añade un docstring para que se guarde qué hace la función en futuras consultas.
- "While True" se utiliza para crear un bucle sin final hasta que se llega a un return. Este return también permitirá usar el dato a posteriori.
- El return de la función sólo aparece hasta que:
 1. El input recibido se convierte a float sin error, y, además:
 2. Se cumple el if, si se ha indicado valores de mínimo y/o máximo
Si no se indica mínimo y/o máximo, la función por defecto aplica "None".
- Usamos un try / except para controlar qué pasa en una situación excepto en otra situación concreta. En nuestro caso, cuando salta el error ValueError envía un mensaje al usuario para volver a entrar el dato. Como no hay ningún "return" dentro de Except, el programa vuelve al principio del "while True" y ejecuta el try de nuevo. Cuando se valida el float y la condición if, termina la función.
- ValueError salta cuando un valor tiene un tipo correcto pero no es válido para realizar nuestra línea de código.

Función 2: "calculo_IMC(weight_user, height_user)"

- realiza un cálculo simple según la fórmula de IMC. Necesita dos parámetros de entrada externos (altura y peso). Retorna el resultado para posterior uso.
-

Función 3: "mensaje_segun_imc(imc_calculado)"

- requiere un parámetro externo para ejecutar una clasificación con los elementos de control if / elif / else. Return permite su uso externo.

Función 4: "main_imc()"

- Llamada de las funciones con la función principal (main):
- "**height_user**" guarda el dato en una variable después de llamar la función "**pedir_medida**" con un mensaje adaptado para la altura, especificado en esa llamada. Lo mismo en el caso de "**weight_user**", adaptado a peso. Se incluyen valores mínimo y máximo para crear un rango a validar en cada una.
- "**imc_calculado**" guarda el resultado de la llamada a ejecución de la función "**calculo_imc**", donde los parámetros de entrada son los valores guardados en "**height_user**" y "**weight_user**".
- "**mensaje**" guarda el resultado de la llamada a la función "**mensaje_según_imc**", donde el parámetro de entrada es el resultado numérico guardado en "**imc_calculado**".
- "**print()**" para mostrar por pantalla un mensaje con un texto fijo y también argumentos calculados anteriormente.
{imc_calculado:.2f}, :.2f es un formateador, e indica que para el argumento imc_calculado, es f (float) y limitamos a exactamente .2 decimales
La f delante de la cadena indica que usaremos un f-string (formatted string), que permite inserir variables directamente dentro del texto usando {}

1.2 CONVERSOR TEMPERATURAS

Selecciona 2 conversores de modo que al introducir una temperatura devuelva, como mínimo, dos conversiones, y de modo que se puedan guardar. Consejo: Intenta validar los datos previamente, para que envíe un mensaje de advertencia si los datos introducidos por el usuario no están en el formato adecuado.

(EXTRA): Piensa una manera de almacenar todas las posibles conversiones en un solo objeto en lugar de escribir muchos if else en función de la temperatura de origen y la temperatura de destino.

```
1 # función validar entrada user (unidades de temperatura concretas)
2 def pedir_validar_unidad(mensaje, lista_unidades):
3     """
4         Solicita al usuario la unidad de Tº y comprueba que sea válida.
5         Repite la petición hasta que la entrada coincida con una de las unidades permitidas.
6
7     Argumentos:
8         mensaje (string) que muestra al usuario para solicitar la unidad
9         lista_unidades (lista) con lista unidades de Tº válidas
10    Return: string con unidad de medida válida
11 """
12
13    while True:
14        unidad_origen = input(mensaje).strip().capitalize()
15        if unidad_origen in lista_unidades:
16            return unidad_origen
17        else:
18            print(f"Error: elige la unidad de Tº entre las siguientes: {', '.join(lista_unidades)}")
19
20 # Función conversión temperatura y guardar resultado en dict
21 def convertir_temperatura(valor, unidad_origen):
22     """
23         Convierte una temperatura desde una unidad de origen a las otras unidades disponibles.
24
25     Argumentos: valor (float) y unidad_origen (str)
26     Return: (dict) diccionario donde keys: unidades de Tº final y values: valores ya convertidos
27     """
28
29     dic_resultados = {}
30
31     if unidad_origen == "Celsius":
32         dic_resultados["Fahrenheit"] = valor * 9/5 + 32
33         dic_resultados["Kelvin"] = valor + 273.15
34
35     elif unidad_origen == "Fahrenheit":
36         dic_resultados["Celsius"] = (valor - 32) * 5/9
37         dic_resultados["Kelvin"] = (valor - 32) * 5/9 + 273.15
38
39     elif unidad_origen == "Kelvin":
40         dic_resultados["Celsius"] = valor - 273.15
41         dic_resultados["Fahrenheit"] = (valor - 273.15) * 9/5 + 32
42
43     return dic_resultados
44
45 def mostrar_resultados(dic_resultado_conversion, temp_usuario, unidad_usuario, repr_unidad_temp):
46     print(f"Temperatura original: {temp_usuario}° {unidad_usuario}, que equivalen a: \n ")
47     for unidad_final, valor_convertido in dic_resultado_conversion.items():
48         print(f" {unidad_final}: {valor_convertido:.2f} {repr_unidad_temp[unidad_final]}")
49
50 def main_conversor_temperaturas():
51     # lista para delimitar entrada unidad temperatura
52     lista_unidades = ["Celsius", "Kelvin", "Fahrenheit"]
53     # diccionario con representación corta unidad de Temperatura
54     repr_unidad_temp = {
55         "Kelvin": "K",
56         "Celsius": "°C",
57         "Fahrenheit": "°F"
58     }
```

```
59     # llamada función validar/input + guardar valores
60     temp_usuario = pedir_medida("Introduce la temperatura. Por ejemplo '36.70'.")
61     unidad_usuario = pedir_validar_unidad(f"Introduce la medida de origen: {', '.join(lista_unidades)}")
62     # llamada función conversión unidades y guardar resulta
63     dic_resultado_conversion = convertir_temperatura(temp_usuario, unidad_usuario)
64     mostrar_resultados(dic_resultado_conversion, temp_usuario, unidad_usuario, repr_unidad_temp)
65
66
67 main_conversor_temperaturas()
68
```

Python

Error: escribe un valor numérico (y un punto, si deseas decimales). Vuelve a probar.
Temperatura original: 100.0° Celsius, que equivalen a:

Fahrenheit: 212.00 °F
Kelvin: 373.15 K

Para este programa creamos 2 funciones auxiliares, 1 principal y reutilizamos 1:

1. "pedir_validar_unidad(mensaje, lista_unidades)"
2. "convertir_temperatura(valor, unidad_origen)"
3. "pedir_medida(mensaje, mínimo=None, máximo=None)" – REUTILIZADA
4. "main_conversor_temperaturas()" - PRINCIPAL

Función 1: "pedir_validar_unidad_de_medida(mensaje, lista_unidades)"

- Para solicitar un dato string al usuario y validar el dato entrado.
- Requiere 2 parámetros externos:
 1. el mensaje que verá el usuario cuando se le pida el dato
 2. la lista de unidades para (a) comparar que el dato entrado se encuentre en la lista y (b) mostrar la lista de opciones cuando salta el mensaje de error.
- Usa un while True, que repetirá el bucle hasta llegar a un return.
- Primero se crea la variable temporal "**unidad_origen**" que guarda la información del input solicitando el dato al usuario, después de eliminar espacios al inicio y al final con .strip(), y de poner la primera letra en mayúscula y el resto en minúscula con .capitalize().
- Dentro del bucle, usamos estructura de control if/else para determinar la acción:
 1. Si la cadena de texto guardada como **unidad_origen** coincide con algun elemento de la lista "**lista_unidades**" (cumple True), se realiza un return de **unidad_origen** y termina el bucle y la función.
 2. En caso contrario (False), lanza un mensaje de error que incluye los valores del listado aceptados gracias al uso de f-string, que permite sustituir valores con la llamada en formato {variable}.
- '**'.'**.join(**lista_unidades**) extrae los elementos de la lista y los une, añadiendo coma y un espacio entre ellos.
 1. Con **.join()** → Celsius, Kelvin, Fahrenheit
 2. Sin **.join()** → ['Celsius', 'Kelvin', 'Fahrenheit']

Función 2: "convertir_temperatura(valor, unidad_origen)"

- Convierte una temperatura desde una unidad de origen a las otras unidades.
- Argumentos: valor (float) y unidad_origen (str).
- Return: diccionario donde keys=unidades de Tª final y values=valores convertidos.
- Primero crea un diccionario vacío donde guardará los resultados de la conversión.
- Con una estructura de control if / elif / else se establece el cálculo a realizar según la unidad_origen y valor introducidos por el usuario. Cada cálculo se guarda como llave:valor del diccionario.

Función 3: "pedir_medida(mensaje, minimo, maximo)" , declarada y explicada en ejercicio calculadora IMC. Solicita valor numérico al usuario y lo valida o vuelve a solicitar.

Función 4: main_conversor_temperaturas() . Incluye:

Declaración de variables:

```
lista_unidades = ["Celsius", "Kelvin", "Fahrenheit"]
```

Se crea la lista para declarar las unidades de medida de temperatura que luego se usarán para comparar con los valores de entrada de la función "pedir_validar_unidad_de_medida(mensaje, lista_unidades)" y validar así los datos. También servirá como lista de valores para el mensaje input y el mensaje de error.

```
repr_unidad_temp = { "Kelvin": "K", "Celsius": "\u00b0C", "Fahrenheit": "\u00b0F"}
```

Se crea diccionario con la representación corta de cada unidad de Temperatura para el print final.

Llamada funciones de validar/input + guardar valores

"temp_usuario" guarda el dato numérico obtenido en la llamada de la función

"pedir_medida" con un mensaje especificado en la llamada y adaptado a la temperatura. Como el valor mínimo y máximo no se indican y en la función se estableció que por defecto fueran "None" , no afectará la ejecución de la función.

"unidad_usuario" , guarda un string de salida. Aquí se establece el mensaje del input, adaptado a la unidad de temperatura que vuelve a usar ", ".join(lista_unidades), para mostrar las únicas opciones admitidas en la función llamada.

Llamada función conversión unidades y guardar resultado

"dic_resultado_conversion" guarda el diccionario de resultados después de llamar la función "convertir_temperatura" introduciendo los argumentos anteriormente guardados (temp_usuario, unidad_usuario).

Mostrar resultados

- El 1er print muestra por pantalla un mensaje fijo y los valores guardados en variables acerca de la temperatura original.
- El 2ndo print muestra el resultado de la conversión. Se inicia con un bucle for que recorre cada par key:value (unidad:valor) del diccionario de resultados. Para poder recorrer cada par de valores se usa el método .items(), que convierte cada key:value en un listado de dos valores de tipo tupla. El bucle for asigna el primer valor de la tupla (llave original) como elemento “**unidad_final**” y el segundo elemento de la tupla (valor) como “**valor_convertido**” . Y para cada conversión guardada en el diccionario de resultados hace un print con la equivalencia. Sin .items() el bucle sólo podría recorrer las llaves.
- La f delante de la cadena indica que usaremos un f-string (formatted string), que permite inserir variables directamente dentro del texto usando {}
- **:.2f** es un formateador. Indica que es f (float) y limitamos la visualización a 2 decimales.
- “repr_unidad_temp[**unidad_final**]” accede al diccionario de equivalencias unidad de medida Tº – símbolo corto, usando como key el contenido de la variable “**unidad_final**” . Como que esta variable contiene el nombre completo i coincide con la llave del diccionario, Python retorna el símbolo correspondiente almacenado como value.

1.3 CONTADOR DE PALABRAS DE UN TEXTO

Escribe una función que, dado un texto, muestre las veces que aparece cada palabra. Intenta que se gestionen todas las casuísticas que hagan que el programa no funcione correctamente.

(EXTRA): ¿Cuál es la longitud media de las palabras del texto que has escrito? "¿Hola cómo va?" debería devolver $(4+3+2) / 3 = 3$

NOTA 1: Se incluye en el archivo ipynb 2 alternativas de código, una usando métodos de las librerías COLLECTIONS y STRING y la otra sin usarlo.

NOTA 2: las mismas palabras con y sin tilde se considerarán diferentes, dado que en español pueden tener diferente significado.

1.3 VERSIÓN SIN COLLECTIONS

```
1 def limpieza_simbolos_y_minusculas(texto):
2     """
3         Convierte el texto en minúscula y elimina símbolos de puntuación
4         Argumento: texto (str) a analizar
5         Return: texto (str) ya limpiado
6     """
7     texto_limpio_minusc = texto.lower()
8     for simbol in ",;:!¿¡()[{}]->...&*\'\"":
9         |     texto_limpio_minusc = texto_limpio_minusc.replace(simbol, "")
10    return texto_limpio_minusc
11
12 def separacion_palabras(texto_limpio_minusc):
13     """
14         Separa las palabras del texto a partir del espacio entre palabras
15         Argumento: texto (str) a procesar
16         Return: lista (list) de palabras
17     """
18     lista_palabras_texto = texto_limpio_minusc.split()
19     return lista_palabras_texto
20
21 def recuento_palabras(lista_palabras_texto):
22     """
23         Cuantas veces aparece una palabra en la lista
24         Argumento: lista de palabras
25         Return: diccionario con palabras únicas como claves y frecuencia como valores
26     """
27     dic_recurrencia_palabras = {}
28     for palabra in lista_palabras_texto:
29         if palabra in dic_recurrencia_palabras:
30             |     dic_recurrencia_palabras[palabra] += 1
31         else:
32             |     dic_recurrencia_palabras[palabra] = 1
33     return dic_recurrencia_palabras
34
35 def pedir_texto(mensaje):
36     """
37         Solicita al usuario un texto.
38         Argumentos: mensaje (string) que se muestra al usuario al solicitar la entrada.
39         Return: string con texto
40     """
41     return input(mensaje)
42
43 def bucle_print(diccionario, mensaje_print):
```

```
44     """
45     Muestra un texto fijo y recorre el diccionario para mostrar cada par de clave:valor en una línea.
46     Argumento: diccionario, mensaje fijo a imprimir (str)
47     Return none (Solo muestra por pantalla)
48     """
49     print(mensaje_print)
50     for clave, valor in diccionario.items():
51         print(f"{clave}: {valor}")
52
53 mensaje = "Introduce el texto que quieras analizar"
54 texto = pedir_texto(mensaje)
55 mensaje_print = "Recuento de palabras en el texto:"
56
57 def main_contador_palabras(texto, mensaje_print):
58     texto_limpio_minusc = limpieza_simbolos_y_minusculas(texto)
59     lista_palabras_texto = separacion_palabras(texto_limpio_minusc)
60     resultado_recuento = recuento_palabras(lista_palabras_texto)
61     bucle_print(resultado_recuento, mensaje_print)
62     return texto
63
64 main_contador_palabras(texto, mensaje_print)
```

✓ 6.7s Python

```
Recuento de palabras en el texto:
hola: 3
va: 2
vamensaje: 1
texto: 1
aquí: 1
```

Para este programa creamos 5 funciones auxiliares + 1 principal:

- "limpieza_simbolos_y_minusculas(texto)"
- "separacion_palabras(texto_limpio_minusc)"
- "recuento_palabras(lista_palabras_texto)"
- "pedir_texto(mensaje)"
- "bucle_print(diccionario, mensaje_print)"
- "main_contador_palabras()"

Función 1: "limpieza_simbolos_y_minusculas(texto)"

- Convierte el texto en minúscula y elimina símbolos de puntuación.
- Argumento: texto (str) a analizar. Return: texto (str) limpio.
- Primero aplica el método `.lower()` al texto de entrada, que transforma toda la palabra en minúscula para que no haya duplicados (Python es case sensitive).
- Realiza un bucle "for" por valor para cada elemento/carácter de la cadena proporcionada (la cadena incluye los símbolos habituales que pueden estar pegados a una palabra por motivos de puntuación). Esto impedirá que el sistema cree palabras duplicadas cuando las palabras son realmente la misma, pero que por tener un símbolo de puntuación serían reconocidas como diferentes. Por ejemplo "Hola!" = "Hola". El método usado es `".replace(valorAntiguo, ValorNuevo, ocurrenciasAcambiar)"`. OcurrenciasAcambiar es opcional y no lo usamos, ya que queremos eliminar todos los símbolos. `.replace(símbolo, "")`. "" = a nada. Guardamos y hacemos return de la variable con el texto reemplazado.

Función 2: "separacion_palabras(texto_limpio_minusc)"

- Separa las palabras del texto a partir del espacio entre palabras.
- Argumento: texto (str) a procesar. Return: lista de palabras.
- Realizamos un bucle "for" por valor. La secuencia sobre la que recorrerá el bucle for es el texto "limpio" de caracteres del anterior bucle al que le aplicamos el método ".split(separador, maxsplit)". Este método separa la cadena de texto entera en cadenas pequeñas (palabras). Por defecto separará las palabras por espacios entre palabras (a menos que se le especifique otro tipo de separador). También se le puede especificar el número de separaciones que debe hacer. Por defecto, si no se escribe nada, separa todas las ocurrencias del separador.

Función 3: "recuento_palabras(lista_palabras_texto)"

- Cuantas veces aparece una palabra en la lista.
- Argumento: lista de palabras.
- Return: diccionario con palabras únicas como claves y frecuencia como valores.
- Primero crea un diccionario vacío e inicia un bucle for para recorrer cada palabra de la lista entrada. Dentro de las acciones del bucle for:
 1. Aplica una estructura de control condicional con if/else. Si la palabra minúscula se encuentra como "key" en el diccionario que hemos creado en la función, entonces le suma "1 ocurrencia".
 2. En caso contrario, declara esa palabra como "key" del diccionario y le asigna 1 ocurrencia inicial.
- Al terminar el bucle, retorna el diccionario lleno de las palabras únicas "key" y las ocurrencias "llaves".

Función 4: "pedir_texto(mensaje)"

- Incluye un input para que solicite el texto al usuario.
- El mensaje a mostrar al usuario en el momento de solicitud se establece como parámetro externo para que se pueda reutilizar mejor la función y no quede limitada por un mensaje concreto no aplicable a otras situaciones.
- Por defecto input devuelve un string, no hay que indicar el tipo de dato.

Función 5: "bucle_print(diccionario, mensaje_print)"

- Print() muestra los resultados por pantalla con un texto fijo y/o variables. "\n" hace un salto de línea. "f" significa f string (formated string) y se usa dentro del print() para incluir variables entre {} para mostrarse dentro del texto fijo.
- El bucle muestra cada par clave:valor en una línea separada.

Función 6: "main_contador_palabras(texto)" . Incluye:

Llamada de las funciones auxiliares

`texto_limpio_minusc = limpieza_simbolos_y_minusculas(texto)` guarda en la variable el resultado del texto limpiado con la función llamada.

`lista_palabras_texto = separacion_palabras(texto_limpio_minusc)` guarda en la variable el resultado de la función llamada para la separación del texto (lista de palabras).

`resultado_reuento = recuento_palabras(lista_palabras_texto)` guarda el resultado de la función que crea el diccionario con el recuento de ocurrencia del listado de palabras.

Print resultados: muestra por pantalla el resultado final

Incluimos un return del texto para poderlo usar en el cálculo promedio longitud de las palabras

Declaración de variables:

`mensaje_usuario = "Introduce el texto que quieras analizar"`

`texto = pedir_texto(mensaje)` guarda el texto entrado por el usuario con la función de solicitud de ingreso del input.

`mensaje_print = "Recuento de palabras en el texto:"`

1.3 VERSIÓN USANDO COLLECTIONS Y STRING

```
1 from collections import Counter
2 import string
3
4 def limpieza_simb_y_minus_con_string(texto):
5     """
6         Convierte el texto en minúscula y elimina símbolos de puntuación
7         Argumento: texto (str) a analizar
8         Return: texto (str) procesado
9     """
10    tabla = str.maketrans("", "", string.punctuation)
11    return texto.lower().translate(tabla)
12
13 def pedir_texto(mensaje):
14     """
15         Sigue al usuario un texto.
16         Argumentos: mensaje (string) que se muestra al usuario al solicitar la entrada.
17         Return: string con texto
18     """
19    return input(mensaje)
20
21 def bucle_print(diccionario, mensaje_print):
22     """
23         Muestra un texto fijo y recorre el diccionario para mostrar cada par de clave:valor en una línea.
24         Argumento: diccionario, mensaje fijo a imprimir (str)
25         Return none (Solo muestra por pantalla)
26     """
27    print(mensaje_print)
28    for clave, valor in diccionario.items():
29        print(f'{clave}: {valor}')
30
31 def main_contador_con_collections(texto_a_analizar, mensaje_print):
32    texto_limpio = limpieza_simb_y_minus_con_string(texto_a_analizar)
33    palabras = texto_limpio.split()
34    resultado = Counter(palabras)
35    bucle_print(resultado, mensaje_print)
```

```
36
37 mensaje_usuario = "Introduce el texto que quieras analizar"
38 texto = pedir_texto(mensaje_usuario)
39 mensaje_print = "Recuento de palabras en el texto:"
40
41
42 main_contador_con_collections(texto, mensaje_print)
```

✓ 12.6s

Python

```
Recuento de palabras en el texto:
hola: 3
text: 2
contador: 1
1: 2
paraules: 1
la: 2
```

Para este programa creamos 3 funciones auxiliares + 1 principal:

- "limpieza_simb_y_minus_con_string(texto):"
- "pedir_texto(mensaje)"
- "bucle_print(resultado)"
- "main_contador_con_collections(texto_a_analizar)"

Se importan 2 módulos:

- Collections.Counter: permite contar automáticamente las ocurrencias de los elementos de un iterable (en este caso, palabras de un texto solicitado al usuario).
- String
 1. punctuation: proporciona una cadena con todos los símbolos de puntuación estándar, evitando tener que escribirlos manualmente.
 2. str.maketrans(x, y, z) crea una tabla de traducción (mapping table) que se usará para traducir.

Función 1: "limpieza_simb_y_minus_con_string(texto)

- X = conjunto de caracteres a reemplazar. También puede ser un diccionario con clave:valor {qué reemplazar : con qué}.
- Y = (opcional) caracteres por los que se reemplazan. Misma largaria que x. Cada carácter de x se reemplaza con el carácter de y, con mismo orden 1-1, 2-2, 3-3...
- Z = (opcional). Caracteres que se deben eliminar del string original.
- Con **str.maketrans("", "", string.punctuation)** no se define ningún reemplazo, pero si la lista de caracteres a eliminar.
- **texto.lower().translate(mapping_table)**, donde:
- **.lower()** transforma el texto a minúsculas para evitar que Python interprete palabras iguales como diferentes por tener minús/mayús (Py es case sensitive).
- **".translate(mapping_table)"** realiza la conversión según lo establecido en la tabla de mapeo anteriormente creada (borrará los símbolos en este caso).

Función 2: "pedir_texto(mensaje)"

- Incluye un input para que solicite el texto al usuario.
- El mensaje a mostrar al usuario en el momento de solicitud se establece como parámetro externo para que se pueda reutilizar mejor la función y no quede limitada por un mensaje concreto no aplicable a otras situaciones.
- Por defecto input devuelve un string, no hay que indicar el tipo de dato.

Función 3: "bucle_print(diccionario, mensaje_print)"

- Print() muestra los resultados por pantalla con un texto fijo y/o variables. "\n" hace un salto de línea. "f" significa f string (formatted string) y se usa dentro del print() para incluir variables entre {} para mostrarse dentro del texto fijo.
- El bucle muestra cada par clave:valor en una línea separada.

Función 4: "main_contador_con_collections(texto_a_analizar)"

-Llamar la función eliminar símbolos de puntuación y mayúsculas:

```
texto_limpio = limpieza_simb_y_minus_con_string(texto_a_analizar)
```

-Aplicar método ".split(separador, maxsplit)" para separar palabras

```
palabras = texto_limpio.split() → por defecto separa por el espacio
```

-Aplicar counter(iterable) que crea un diccionario directamente donde las claves son las palabras únicas y los valores el nombre de apariciones.

```
resultado = Counter(palabras)
```

-Llamar función print

```
bucle_print(resultado, mensaje_print)
```

Declaración de variables.

- mensaje_usuario: el texto a mostrar al usuario con el input
- mensaje_print: el mensaje para el print de este programa
- Se llama a la función "pedir_texto(mensaje)" y se guarda externamente para que se pueda acceder con el siguiente ejercicio y cualquier otra tarea donde sea necesario el texto original.

1.3 EXTRA: CÁLCULO DEL PROMEDIO DE LONGITUD DE LAS PALABRAS

```
1 # EXERCICIO EXTRA: cálculo promedio longitud de las palabras
2
3 def promedio_long_palabras(lista_palabras_texto):
4     """
5         Calcula el promedio de letras por palabra.
6         Argumento: lista de palabras a analizar (list)
7         Return: promedio (float) de letras por palabras, redondeado a 2 decimales
8     """
9     if len(lista_palabras_texto) == 0: #si lista vacía, evita error division por 0
10        return 0
11    total_letras = 0
12    for palabra in lista_palabras_texto:
13        total_letras += len(palabra)
14    return round(total_letras / len(lista_palabras_texto),2)
15
16 def main_longitud_palabras(texto):
17     # llamada funciones procesado
18     texto_limpio_minusc = limpieza_simbolos_y_minusculas(texto)
19     lista_palabras_texto = separacion_palabras(texto_limpio_minusc)
20     longitud_texto = promedio_long_palabras(lista_palabras_texto)
21     # Muestra resultados:
22     print(f"El promedio de la longitud de las palabras del texto: {longitud_texto}")
23
24 main_longitud_palabras(texto)
```

✓ 0.0s

Python

El promedio de la longitud de las palabras del texto: 4.25

Para este programa creamos 1 función auxiliar, 1 principal y aprovechamos 2 auxiliares anteriores:

- "limpieza_simbolos_y_minusculas(texto)" - REUTILIZADA
- "separacion_palabras(texto_limpio_minusc)" – REUTILIZADA
- "promedio_long_palabras(lista_palabras_texto)"
- "main_longitud_palabras()"

Función 1: "promedio_long_palabras(lista_palabras_texto)

- Calcula el promedio de letras por palabra.
- Argumento: lista de palabras a analizar (list)
- Retorno: promedio (float) de letras por palabras, redondeado a 2 decimales
- Con una estructura de control if al inicio, indicamos que si la lista está vacía, es decir, cantidad de palabras = 0 → (len(lista)), retorna 0 y termina la función. Esto nos permite evitar que salte un error de división por 0 y también permitirá usar el valor 0 como respuesta del print (tiene lógica).
- A continuación se declara la variable total_letras = 0 que servirá como acumulador.
- Inicio del bucle for para recorrer cada palabra del listado de palabras introducido como argumento.
- Len(palabra) recuenta los caracteres de cada palabra y total_letras += suma el valor que había en la variable más el nuevo número de la iteración.

- Al terminar el bucle, se retorna un valor que proviene del cálculo de dividir total_letras de la lista con el total palabras de la lista. El round(valor, 2) redondea la salida a 2 decimales.

Función 2: “main_longitud_palabras()” . Incluye:

Llamada de las funciones:

- “limpieza_simbolos_y_minusculas(texto)”
- “separacion_palabras(texto_limpio_minusc)”
- “promedio_long_palabras(lista_palabras_texto)”

Mostrar resultados:

Print() muestra los resultados por pantalla, con una cadena de texto fijo y una parte variable gracias a “f” (formated string), que permite incluir las variables con {} dentro del texto string.

1.4 INVERTIR UN DICCIONARIO

Intercambia las claves y valores de un diccionario. Los valores y claves en el diccionario original deberían ser únicos pero, en su defecto, la función debe imprimir un mensaje de advertencia, junto con una lista con los valores asociados a la clave repetida.

```
1 def inversion_diccionario(dic_original):
2     """
3         Intercambio de llaves por valores y viceversa de un diccionario dado.
4         Si encuentra valores duplicados en el diccionario original,
5         muestra un mensaje de alerta y un listado del valor y las llaves donde está duplicado.
6
7         Argumento: diccionario original
8         Return: nuevo diccionario invertido
9         """
10
11    dic_invertido = {}
12    dic_llaves_duplicadas = {}
13    for key, value in dic_original.items():
14        if value not in dic_invertido:
15            dic_invertido[value] = key
16        else:
17            if value not in dic_llaves_duplicadas:
18                dic_llaves_duplicadas[value] = [dic_invertido[value]]
19            dic_llaves_duplicadas[value].append(key)
20    if dic_llaves_duplicadas:
21        print("ALERTA: hay valores duplicados en el diccionario original que impiden crear todos los regis")
22        for value, key in dic_llaves_duplicadas.items():
23            print(f"Valor '{value}' repetido en las siguientes claves: {key}")
24    return dic_invertido
25
26 def main_inversion_dic(dic_a_analizar):
27     dic_final_invertido = inversion_diccionario(dic_a_analizar)
28     print("Diccionario invertido:", dic_final_invertido)
29
30 diccionario_ejemplo = {"a": "león", "b": "ciervo", "c": "caballo", "d": "caballo"}
31 main_inversion_dic(diccionario_ejemplo)
32
```

Python

ALERTA: hay valores duplicados en el diccionario original que impiden crear todos los registros en el diccionario.
Valor 'caballo' repetido en las siguientes claves: ['c', 'd']
Diccionario invertido: {'león': 'a', 'ciervo': 'b', 'caballo': 'c'}

Para este programa creamos 1 función auxiliar + 1 principal.

Función 1: "inversion_diccionario(dic_original)"

- Realiza un intercambio de llaves por valores y viceversa. Si encuentra valores duplicados en el diccionario original, muestra un mensaje de alerta además de un listado con el valor y las llaves donde está duplicado.
- Argumento: diccionario original
- Return: nuevo diccionario invertido
- Creamos un diccionario vacío donde se guardará el diccionario invertido (valores como claves y claves como valores).
- Creamos otro diccionario vacío para almacenar los valores duplicados del diccionario original y las claves asociadas a esos valores.
- Se inicia un bucle for por valor que recorre cada par key:value del diccionario original usando el método .items(), que convierte cada key:value en una tupla con

dos valores y permite recorrer a la vez las claves y los valores. Sin ello el bucle sólo devolvería las llaves.

- Con la estructura de control “**If value not in dic_invertido**” , comprueba si el valor original ya existe como llave en el diccionario invertido, y si no, crea el par key:value en el mismo, pero con la llave original como valor y el valor como llave.
- En caso de que ya exista, significa que hay un valor duplicado. Entonces la siguiente estructura de control “if” mira si el par ya existe en el diccionario de duplicados. En caso que no, crea un nuevo registro en el diccionario dic_llaves_duplicadas y le asigna como key el valor duplicado. Para el valor se crea una lista con la llave original duplicada.
- Si la llave duplicada ya existe, entonces se accede a la lista creada y añade una nueva clave duplicada con “.append()” : dic_llaves_duplicadas[value].append(key)
- if dic_llaves_duplicadas = si el diccionario no está vacío, se hace un print() con un mensaje de alerta fijo, y un print dentro de un bucle for que recorre cada par de llave:valor del dic_llaves_duplicadas gracias a “.items()” que convierte el par en una tupla de 2 valores y así otro print dentro del bucle imprime por pantalla donde se encuentran los duplicados de cada valor:llave.

Función 2: “main_inversion_dic(dic_a_analizar)”

Dic_final_invertido guarda el diccionario resultado de llamar a la función de inversión del diccionario, que toma por argumento un dic_a_analizar.

Declaración de variables:

Un diccionario test del programa (diccionario_ejemplo).

NIVEL 2

2.1 CONTADOR Y ORGANIZADOR DE PALABRAS DE UN TEXTO

2.1 VERSIÓN SIN COLLECTIONS

```
1 # Subir fichero y transformar a string:
2 def subir_file_pasar_txt_str(link_fichero):
3     """
4         Abre el fichero en modo lectura (r) y lo convierte a string. Programa el cierre automático.
5         Avisa con mensaje cuando archivo no encontrado.
6
7         Argumento: ruta del fichero.
8         Return: fichero en texto (str)
9         """
10    try:
11        with open(link_fichero, "r") as fichero:
12            return fichero.read()
13    except FileNotFoundError as e:
14        print("Error: no se encuentra el archivo", e)
15        return ""
16
17 # crear dicc dentro de dic con formato {1a letra: {palabras:recuento}}
18 def agrupar_1a_letra(diccionario):
19     """
20         Agrupa las palabras por su primera letra, creando un subdiccionario como {letra: {palabra: recuento}}.
21         Argumento: diccionario
22         Return diccionario con agrupación por la primera letra
23         """
24     dic_agrupado_1a_letra = {}
25     for palabra_clave, valor in diccionario.items():
26         if not palabra_clave:
27             continue # evita IndexError si llave está vacía
28         primera_letra = palabra_clave[0]
29         if primera_letra not in dic_agrupado_1a_letra:
30             dic_agrupado_1a_letra[primera_letra] = {}
31             dic_agrupado_1a_letra[primera_letra][palabra_clave] = valor
32     return dic_agrupado_1a_letra
33
34 # Ordenar palabras y transformar a diccionario
35 def ordenar_palabras_int_ext(diccionario):
36     """
37         Ordena alfabéticamente el diccionario externo (letras iniciales) y los diccionarios internos (palabras).
38         Argumento: diccionario.
39         Return: diccionario ordenado
40         """
41
42     diccionario_ordenado = {}
43     for primera_letra, subdic in sorted(diccionario.items()):
44         diccionario_ordenado[primera_letra] = dict(sorted(subdic.items())) #dict= crear dic y no lista de tuplas
45     return diccionario_ordenado
46
47
48 def main_contar_ordenar_texto(enlace):
49     # Ejecutamos subida archivo y conversión a string
50     texto_blanca = subir_file_pasar_txt_str(enlace)
51     # Ejecutamos limpieza y recuento palabras (crea dic con "palabra:repeticiones")
52     texto_sin_simbolos_minus = limpieza_simbolos_y_minusculas(texto_blanca)
53     lista_palabras_separadas = separacion_palabras(texto_sin_simbolos_minus)
54     dic_con_recuento = recuento_palabras(lista_palabras_separadas)
55     # Ejecutamos crear dic con formato 1 letra + subdic:
56     diccionario_agrupado = agrupar_1a_letra(dic_con_recuento)
57     # Ejecutamos ordenación diccionario
58     dic_ordenado = ordenar_palabras_int_ext(diccionario_agrupado)
59     # Print en vertical:
60     for clave_externa, subdiccionario in dic_ordenado.items(): #items() para mostrar tmb value
61         print(f'{clave_externa.upper()}:')
62         for clave_interna, valor in subdiccionario.items():
63             print(f" {clave_interna}: {valor}")
64     return dic_ordenado
65
66 main_contar_ordenar_texto(r"C:\Users\vanes\Documents\CURSOS\IT Academy - Anàlisis de dades\ESPECIALITAT\Sprint 9 Pyth
```

```
66 main_contar_ordenar_texto(r"C:\Users\vanes\Documents\CURSOS\IT Academy - Anàlisis de dades\_ESPECIALITAT\Sprint 9 Pyth
A:
a: 3
agua: 1
al: 2
alba: 4
alcobas: 1
alimenta: 1
```

Para este programa creamos 3 funciones auxiliares, 1 principal y reutilizamos 3:

- "subir_file_pasar_txt_str(link_carpeta)"
Lee un fichero de texto desde una ruta indicada y devuelve su contenido como una cadena de texto (str). Avisa con mensaje cuando archivo no encontrado.
- "limpieza_simbolos_y_minusculas(texto)" - REUTILIZADA
Limpia el texto eliminando signos de puntuación y pasando todas las palabras a minúscula.
- "separacion_palabras(texto_limpio_minusc)" - REUTILIZADA
Separa el texto en palabras individuales utilizando los espacios como delimitadores y devuelve una lista de palabras.
- "recuento_palabras(lista_palabras_texto)" - REUTILIZADA
Calcula la frecuencia de aparición de cada palabra y devuelve un diccionario con el formato {palabra: recuento}.
- "agrupar_1a_letra(diccionario)"
Agrupa las palabras por su primera letra, creando un subdiccionario como valor con el formato {letra: {palabra: recuento}}.
- "ordenar_palabras_int_ext(diccionario)"
Ordena alfabéticamente el diccionario externo (letras iniciales) y los diccionarios internos (palabras).
- "main_contar_ordenar_texto(enlace)"

Función 1: subir_file_pasar_txt_str(link_carpeta)

- Usamos un **try / except** para controlar qué pasa en una situación excepto en otra situación concreta. En este caso, cuando no se encuentra el archivo y salta **FileNotFoundException**, se muestra un mensaje al usuario avisando + e (alias asignado al error), para que muestre el mensaje + los detalles del error. Como hay "return" dentro de Except, el programa termina.
- **open(rutaArchivo, modo[opc]):** abre el archivo y lo devuelve como objeto. El "modo" es opcional, r = read (abre en modo lectura). También existen otros modos: w (write = modo lectura, y crea archivo si no existe); a (append = abre para añadir y crea el archivo si no existe); x (crea el archivo y da error si ya existe).

- Añadir “**with**” al “**open()**” automatiza el cierre del archivo. En caso contrario debería hacerse de forma manual con nombre VariableArchivo.**close()** al terminar la función.
- Cuando se llame a la función, se pondrá **r'** delante del link para modificar la contrabarra de la ruta automáticamente y no tener que cambiar a mano las barras a la derecha.

Función 2: agrupar_1a_letra(diccionario)

- Agrupa las palabras por su primera letra, creando un subdiccionario como valor con el formato `{letra: {palabra: recuento}}`.
- Se crea un diccionario vacío donde agruparemos.
- Se realiza un bucle “**for**” para recorrer la llave y valor del diccionario de entrada gracias al método `.items()` que lo transforma en una tupla de dos valores para cada iteración. Sin esto el bucle sólo podría acceder a la llave.
- Para cada iteración, una estructura de control `if not palabra_clave: continue` ayuda a evitar un error que pare el proceso si una llave del diccionario está vacía como “`''` o `None`. Esto permite saltar la iteración y no detener el proceso. Los diccionarios si pueden tener una llave vacía como “`''` porque se considera un string válido.
- Habiendo pasado la estructura de control de este error, se declara la variable “`primera_letra`” que guarda la primera letra de la clave extrayendo con `“key[0]”` (el carácter en la primera posición de la clave).
- El siguiente `if` indica que, si la primera letra no se encuentra en el diccionario creado en la función, se asigna la letra como clave del diccionario y se declara el valor como un diccionario vacío.
- Cuando ya existe la letra como clave en ese diccionario, se inserta como valor la palabra:recuento accediendo al subdiccionario con esta metodología:
`“diccionario_externo[clave_externa][clave_interna] = valor”`
- Se termina con un `return` del diccionario creado y llenado en la función.

Función 3: ordenar_palabras_int_ext(diccionario)

- Ordena alfabéticamente el diccionario externo (letras iniciales) y los diccionarios internos (palabras).
- *** Crea un diccionario vacío donde se almacenarán los datos ordenados.**
- Se realiza un bucle `for` del diccionario de entrada.
- `“.items()”` retorna todos los pares de clave:valor del diccionario en formato tupla de dos valores para que el bucle puede recorrer ambos datos.
- Con `“.sorted()”` delante de la tupla del diccionario, se ordena alfabéticamente.

- Como acción dentro del bucle, se asigna la clave al nuevo diccionario y el valor del subdiccionario. Antes de copiar la clave al nuevo diccionario, se aplica un .items al subdiccionario, se ordena con otro .sorted() y se aplica “dict()” para transformar esa tupla ordenada de nuevo a formato diccionario key:value.
- Termina retornando el nuevo diccionario ordenado.

* Para sustituir el diccionario original se debería cambiar el return por estas 3 líneas:

```
# diccionario.clear()  
# diccionario.update(diccionario_ordenado)  
# return diccionario
```

Función 4: main_contar_ordenar_texto(enlace). Incluye:

Declaración de variables y llamada de las funciones:

Se realiza en aquellas funciones que tienen un return, para que puedan ejecutarse:

- Ejecutamos subida archivo y conversión a string
- Ejecutamos limpieza y recuento palabras (crea dic con "palabra:repeticiones")
- Ejecutamos crear dic con formato 1 letra + subdic:
- Ejecutamos ordenación diccionario

Mostrar resultados

Se termina con un bucle for que recorre el diccionario final ordenado con .items() y realiza 2 acciones:

- 1 print de la clave externa (letra) a la que aplica “.upper()” para visualizarla en mayúscula.
- Otro bucle para recorrer el subdiccionario para poder hacer el print de la clave interna y su valor.

2.1 VERSIÓN USANDO COLLECTIONS

```
1  from collections import Counter, defaultdict
2
3  def subir_file_pasar_txt_str(link_archivo):
4      """
5          Abre el fichero en modo lectura (r) y lo convierte a string. Programa el cierre automático.
6          Avisa con mensaje cuando archivo no encontrado.
7
8          Argumento: ruta del fichero.
9          Return: fichero en texto (str)
10         """
11        try:
12            with open(link_archivo, "r") as fichero:
13                return fichero.read()
14        except FileNotFoundError as e:
15            print("Error: no se encuentra el archivo", e)
16            return ""
17
18    # Agrupar palabras por primera letra usando defaultdict(Counter)
19    """
20        Agrupa las palabras por su primera letra, creando un subdiccionario como {letra: {palabra: recuento}}.
21        Argumento: diccionario
22        Return diccionario con agrupación por la primera letra
23        """
24    def agrupar_1a_letra(diccionario):
25        dic_agrupado = defaultdict(Counter)
26        for palabra, recuento in diccionario.items():
27            if palabra: # evita clave vacía
28                dic_agrupado[palabra[0]][palabra] = recuento
29        return dic_agrupado
30
31    # Ordenar diccionarios interno y externo
32    def ordenar_palabras_int_ext(diccionario):
33        diccionario_ordenado = {}
34        for letra, subdic in sorted(diccionario.items()):
35            diccionario_ordenado[letra] = dict(sorted(subdic.items()))
36        return diccionario_ordenado
37
38    def main_contar_ordenar_texto(enlace):
39        texto = subir_file_pasar_txt_str(enlace)
40        texto_limpio = limpieza_simbolos_y_minusculas(texto)
41        lista_palabras = separacion_palabras(texto_limpio)
42        dic_con_recuento = Counter(lista_palabras)
43        dic_agrupado = agrupar_1a_letra(dic_con_recuento)
44        dic_ordenado = ordenar_palabras_int_ext(dic_agrupado)
45        for letra, subdic in dic_ordenado.items():
46            print(f'{letra.upper()}:')
47            for palabra, recuento in subdic.items():
48                print(f'  {palabra}: {recuento}')
49        return dic_ordenado
50
51 main_contar_ordenar_texto(r"C:\Users\vanes\Documents\CURSOS\IT Academy - Anàlisis de dades\ESPECIALITAT\Sprint 9 Python\tu me c
52
✓ 0.0s
Python
```

A:
a: 3

Cambios:**1) Función agrupar_1a_letra(diccionario)**

defaultdict(Counter) crea automáticamente un diccionario especial:

- un diccionario externo para cada letra cuyo valor por defecto es un Counter
- el Counter interno será un contador listo para almacenar palabras y recuentos. Evita tener que comprobar si la clave existe antes de insertar.

if palabra:

- Comprueba que la palabra no esté vacía ("" o None).
- Si lo está, se omite la iteración.
- Así evita errores al acceder al primer carácter con palabra[0].

dic_agrupado[palabra[0]][palabra] = recuento

- palabra[0] obtiene la primera letra de la palabra y se usa como clave del diccionario externo.
- Dentro del Counter asociado a esa letra, se guarda la palabra completa con su recuento.

2) Función main_contar_ordenar_texto(enlace)

Antes se utilizaba la función recuento_palabras(). Ahora se sustituye por Counter.

- Counter(lista_palabras) recorre internamente la lista y cuenta cuántas veces aparece cada palabra sin usar bucles ni estructuras if.
- El resultado es un objeto de tipo Counter, clase de collections
- Se comporta como un diccionario y puede usarse directamente en funciones que esperan un diccionario.

2.2 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 segunda, donde estarán los elementos que no pudieron convertirse.

```
1 # Transformar lista variada con tuplas y listas a lista simple:
2 def simplificar_lista_mixta_con_iterables(lista_externa):
3     """
4         Recorre la lista y devuelve una nueva lista plana con elementos individuales, eliminando los elementos iterables.
5
6         Argumento: lista_externa (list) que puede contener tuplas, listas o elementos simples.
7         Return: (list) nueva lista simplificada.
8     """
9     lista_simplificada = []
10    for elemento in lista_externa:
11        if isinstance(elemento, (list, tuple)):
12            lista_simplificada.extend(simplificar_lista_mixta_con_iterables(elemento))
13
14        else:
15            lista_simplificada.append(elemento)
16    return lista_simplificada
17
18 # Separar elementos según si son convertibles a float:
19 def creacion_listas_float_no_float(lista_externa):
20     """
21         Clasifica los elementos de una lista según si son convertibles a float.
22
23         Argumento: lista_externa que puede contener elementos de distintos tipos.
24         Return:
25             - lista 1: elementos no convertibles a float.
26             - lista 2: elementos convertidos a float.
27     """
28
29     elem_float_ok = []
30     elem_no_float = []
31     for objeto in lista_externa:
32         try:
33             elem_float_ok.append(float(objeto))
34         except (ValueError, TypeError):
35             elem_no_float.append(objeto)
36     return elem_no_float, elem_float_ok
37
38 def main_conversion_datos(lista_externa):
39     """
40         función principal que coordina el flujo del programa
41         Argumento:
42             Return: listas separadas con elementos convertidos a float y los que no
43     """
44     lista_ready_para_float = simplificar_lista_mixta_con_iterables(lista_externa)
45     lista_elem_no_float, lista_float_ok = creacion_listas_float_no_float(lista_ready_para_float)
46     print("Lista de elementos convertidos a float:", lista_float_ok)
47     print("Lista de elementos NO convertibles a float:", lista_elem_no_float)
48
49 main_conversion_datos(['1.3', 'one' , '1e10' , 'seven', '3-1/2', ('2',1,1.4,'not-a-number'), [1,2,'3',3.4]
✓ 0.0s
```

Lista de elementos convertidos a float: [1.3, 1000000000.0, 2.0, 1.0, 1.4, 1.0, 2.0, 3.0, 3.4]

Lista de elementos NO convertibles a float: ['one', 'seven', '3-1/2', 'not-a-number']

(['one', 'seven', '3-1/2', 'not-a-number'],
[1.3, 1000000000.0, 2.0, 1.0, 1.4, 1.0, 2.0, 3.0, 3.4])

Para este programa creamos 2 funciones auxiliares + 1 principal:

- "simplificar_lista_mixta_con_iterables(lista_externa)"
- "creacion_listas_float_no_float(lista_externa)"
- "main_conversion_datos(lista_externa)"

Función 1: simplificar_lista_mixta_con_iterables(lista_externa)

- Comprueba si la lista contiene listas o tuplas anidadas y las extrae para formar una lista con sólo objetos individuales.
- Crea una lista vacía llamada lista_simplificada, donde se irán almacenando los elementos individuales.
- Se ejecuta un bucle for que recorre cada elemento de la lista de entrada.
- Para cada elemento de la iteración se utiliza una estructura de control if + isinstance() que comprueba si el elemento es una tupla o lista:
"isinstance(object, type)".
- Si el resultado es True, llama recursivamente a la misma función para simplificar ese iterable interno.
- Cuando se vuelve a aplicar la función de forma recursiva, esa lista o tupla anidada es recorrida por el bucle por cada elemento individual, y al ser individual pasa a la lista con .append(). El resultado de la llamada recursiva es esa lista sin estructura anidada, cuyos elementos se añaden a la lista_simplificada usando "**".extend()**" que permite añadir varios objetos al final de una lista ("**".append()**" sólo 1).
- Else: cuando el elemento no es una tupla/lista, se realiza "**".append()**" que añade directamente el objeto individual a la lista.
- La función termina retornando la lista simplificada.

Función 2: creacion_listas_float_no_float(lista_externa)

- Se crean 2 listas vacías:
 1. elem_float_ok: almacenará los elementos convertidos a float.
 2. elem_no_float: almacenará elementos que no pueden convertirse a float.
- Se realiza un bucle for, y para cada elemento de la lista iterado se aplican unas acciones dentro de un Try / except, para evitar que el programa se interrumpa si aparece uno de los errores especificados.
- Try: intenta convertir el elemento con "float(objeto)" . Si lo consigue, se añade a la lista elem_float_ok con ".append()" .
- En caso que no se pueda convertir con éxito, si se da uno de estos errores se realiza el .append(objeto) a la lista de elementos "elem_no_float" .
- ValueError = "valor no válido" . El tipo de dato es correcto, pero no puede usarse para la operación solicitada. Por ejemplo "5" o "abc" .
- TypeError = "tipo incompatible con la operación (ej. listas, diccionarios, None)".
- La función retorna una tupla de dos listas, que habrá que desempaquetar al llamar la función a la vez que guardarlas en dos variables. Es tupla por defecto porque Python agrupa automáticamente múltiples valores de retorno en una única estructura. Podría forzar que fuera una lista de listas añadiendo return [lista_1, lista_2], pero las tuplas expresan mejor un paquete de resultados de estructura fija.

Función 3: "main_conversion_datos(lista_externa)" . Incluye:

Declaración de variables y llamada de las funciones:

- Llamada a la función simplificar_lista_mixta_con_iterables(lista_externa) con un ejemplo de lista directamente. Se guarda la lista plana del retorno en una variable.
- Llamada a la función creacion_listas_float_no_float(lista_externa). Se guarda en dos variables el return de la tupla con las dos listas.

Mostrar resultados

Para la comprobación se realizan dos print() con texto fijo y la variable que recoge cada lista (elementos convertidos a float, y los no convertidos).

Retorna 2 listas con los elementos separados convertidos a float y los no convertidos.

NIVEL 3

3.1 GENERADOR DE CONTRASEÑAS

El cliente nos ha encargado una función de Python que genere contraseñas seguras. La función debe depender de los siguientes parámetros: longitud (int): Longitud de la contraseña mayúsculas (bool = True): Si tienen que aparecer mayúsculas minúsculas (bool = True): Si tienen que aparecer minúsculas números (bool = True): Si tienen que aparecer números signos (bool = False): Si tienen que aparecer caracteres especiales (,-\$? o similares) Así pues, si ejecutamos la función de la siguiente forma:

crear_contraseña(10, True, True, True, True)
el output que se tiene que poder guardar debería ser del estilo
"9Er,5Vn8P\$"

(EXTRA) Explora cómo podríamos hacer que la función copiara la contraseña automáticamente en el portapapeles del ordenador (como si la hubiéramos seleccionado y hecho ctrl+copy).

```
1 import numpy as np
2 import string
3 import pyperclip
4
5 # Función crear contraseña
6 def crear_password(longitud, mayusc, minusc, numeros, signos):
7     """
8         Genera contraseñas aleatorias con los criterios de caracteres especificados como argumentos.
9
10    Argumentos:
11        - longitud (int): Número de caracteres de la contraseña.
12        - mayusc (bool): Incluye letras mayúsculas si es True.
13        - minusc (bool): Incluye letras minúsculas si es True.
14        - numeros (bool): Incluye del 0-9 si es True.
15        - signos (bool): Incluye símbolos especiales si es True.
16    Return: (str) cadena de caracteres con la contraseña
17    """
18
19    lista_caracteres = "" # crea cadena vacía
20
21    if mayusc: # comprueba si True (hay)
22        lista_caracteres += string.ascii_uppercase
23    if minusc:
24        lista_caracteres += string.ascii_lowercase
25    if numeros:
26        lista_caracteres += string.digits
27    if signos:
28        lista_caracteres += string.punctuation
29    if not lista_caracteres:
30        raise ValueError("Selecciona como mínimo un tipo de caracteres para generar la contraseña")
31
32    password = ''.join(np.random.choice(list(lista_caracteres), size=longitud))
33    return password
34
35 def main_crear_contraseña():
36     # Ejecutamos crear contraseña
37     password_generado = crear_password(10, True, True, True, True)
38     print(password_generado)
39     # EXTRA- función para copiar automáticamente el texto
40     pyperclip.copy(password_generado)
41
42 main_crear_contraseña()
```

Python

9&~1]r#1%j

Para este programa creamos 1 función auxiliar, 1 principal e importamos 3 librerías:

- "crear_password(longitud, mayusc, minusc, numeros, signos)"
- "main_crear_contraseña()"

Librerías:

- NumPy: orientada a cálculo numérico y tratamiento eficiente de datos numéricos.
- String: proporciona constantes y herramientas útiles para trabajar con texto. En este caso nos es útil para **acceder a conjuntos de caracteres predefinidos** para que el generador de contraseñas tenga una lista completa de todos los caracteres que deben estar incluidos en la contraseña (mayusc, minusc, dígitos, símbolos).
- Pyperclip permite **copiar y pegar texto dentro del sistema** (clipboard)

Función 1: "crear_password(longitud, mayusc, minusc, numeros, signos)"

- La función sirve para generar contraseñas aleatorias con los criterios de caracteres especificados como argumentos según como se declara en la función:
 1. Longitud: número de caracteres de la contraseña
 2. Mayusc, minusc, números, signos: serán valores booleanos que deberán especificarse como True / False en la llamada de la función, para que sean incluidos en las acciones de la función (creación de la contraseña).
- Primero se crea una variable que es una cadena vacía y que servirá como acumulador de los caracteres indicados para la contraseña.
- Se inician las estructuras de control "if" . Cada una lee si su argumento es True/False. En caso de true, la acción es actualizar la variable cadena con los elementos que contenía hasta ese momento + el listado de caracteres que ha sido indicado como True. Para indicar el listado de caracteres se usa el módulo string y su método correspondiente, que tiene conjuntos de caracteres predefinidos:
 1. string.ascii_lowercase → letras minúsculas
 2. string.ascii_uppercase → letras mayúsculas
 3. string.digits → números (0–9)
 4. string.punctuation → símbolos (!@#\$...)
- Para sumar lo que ya había +lo nuevo, se usa "`+ =`" en vez de repetir lo que había. Ej: `lista_caracteres = lista_caracteres + 2` → es lo mismo que: `lista_caracteres += 2`
- Si al final de la función la lista_caracteres sigue vacía (if not), lanza una excepción de ValueError (valor no válido) junto a un mensaje de alerta para el usuario para evitar proseguir y generar una contraseña sin los requisitos suficientes.
- Una vez la cadena incluye los caracteres elegidos sobre los que crear password:
 1. la cadena se transforma a lista con `list()`
 2. se realiza una selección aleatoria de entre los caracteres con `np.random.choice(listado, size = longitud)` donde longitud es argumento
 3. los caracteres seleccionados aleatoriamente se unen con `"" join()`
 4. Esa variable guardada se retorna y termina la función.

Función 2: "main_crear_contraseña()". Incluye:

Declaración de variables, llamada de las funciones y muestra del resultado

Se llama a la función y se guarda la contraseña en una variable y luego se hace print() para mostrar la contraseña al usuario.

3.1 EXTRA: FUNCIÓN PARA COPIAR AUTOMÁTICAMENTE EL TEXTO

Llamamos directamente a la librería pyperclip con su función ".copy():"

```
pyperclip.copy(password_generado)
```

3.2: PROCESADO DE DATOS SIMPLE

Fichero con histórico de partidos de fútbol, con nombres de equipos y resultados. Necesita que procesemos los datos de forma automática, para extraer los resultados que necesita. Utiliza el archivo "historic_partits.txt"

El programa debe devolver:

- El número total de goles que ha marcado cada equipo.
- El nombre del equipo más goleador.
- El nombre del equipo más goleado.
- La clasificación global (cada victoria: 3 pts, empate 1 pts, derrota 0 pts).

```
1 import re
2
3 # Abrir fichero y transformar a lista de strings:
4 def abrir_pasar_file_lista_lineas(link_carpet):
5     """
6         Abre el fichero en modo lectura (r) y lo convierte a una lista de líneas.
7         Programa el cierre automático.
8         Avisa con mensaje cuando no se encuentra el archivo.
9
10        Argumento: ruta del fichero.
11        Return: lista de líneas (list)
12    """
13    try:
14        with open(link_carpet, "r") as fichero:
15            return fichero.readlines()
16    except FileNotFoundError as e:
17        print("Error: no se encuentra el archivo", e)
18        return []
19
20 # Diccionarios para guardar info
21 def crear_diccionarios():
22     """
23         Crea los diccionarios vacíos necesarios para acumular las estadísticas.
24         Return: tres diccionarios vacíos (dict)
25     """
26     dic_goles_hechos = {}
27     dic_goles_recibidos = {}
28     dic_puntos = {}
29     return dic_goles_hechos, dic_goles_recibidos, dic_puntos
30
31 def procesar_partido(linea, dic_goles_hechos, dic_goles_recibidos, dic_puntos):
32     """
33         Procesa la línea del fichero que representa un partido.
34         Ignora la linea si está vacía o tiene más palabras que los datos esperados (3).
35         Extrae los nombres de equipo visitante y local, los goles visitante y local, y
36         calcula los puntos de cada equipo según si es victoria, derrota o empate.
37         Con estos datos actualiza los diccionarios de entrada.
38
39         Argumentos:
40         - linea (str): línea del archivo que contiene la información de un partido.
41         - dic_goles_hechos (dict): diccionario con los goles marcados por cada equipo.
42         - dic_goles_recibidos (dict): diccionario con los goles recibidos por cada equipo.
43         - dic_puntos (dict): diccionario con los dic_puntos obtenidos por cada equipo.
44
45         Return: sin return, actualiza los diccionarios de entrada.
46     """
47
48     linea = linea.strip()
49     if linea == "":
50         return
51
52     datos_diferentes_de_un_partido = [x.strip() for x in re.split("\t+", linea)]
53     if len(datos_diferentes_de_un_partido) != 3:
54         return
55
56     nom_equipo_local, resultado_goles, nom_equipo_visitante = datos_diferentes_de_un_partido
57     try:
58         goles_local, goles_visitante = map(int, resultado_goles.split("-"))
59     except ValueError:
60         return
61
62     # Actualizar diccionario con goles marcados
63     dic_goles_hechos[nom_equipo_local] = dic_goles_hechos.get(nom_equipo_local, 0) + goles_local
64     dic_goles_hechos[nom_equipo_visitante] = dic_goles_hechos.get(nom_equipo_visitante, 0) + goles_visitante
```

```
66     # Actualizar diccionario con goles recibidos
67     dic_goles_recibidos[nom_equipo_local] = dic_goles_recibidos.get(nom_equipo_local, 0) + goles_visitante
68     dic_goles_recibidos[nom_equipo_visitante] = dic_goles_recibidos.get(nom_equipo_visitante, 0) + goles_local
69
70     # Añadir puntos según clasificación
71     if goles_local > goles_visitante:
72         dic_puntos[nom_equipo_local] = dic_puntos.get(nom_equipo_local, 0) + 3
73     elif goles_local < goles_visitante:
74         dic_puntos[nom_equipo_visitante] = dic_puntos.get(nom_equipo_visitante, 0) + 3
75     else:
76         dic_puntos[nom_equipo_local] = dic_puntos.get(nom_equipo_local, 0) + 1
77         dic_puntos[nom_equipo_visitante] = dic_puntos.get(nom_equipo_visitante, 0) + 1
78
79
80 # Cálculo de las métricas solicitadas
81 def calcular_metricas_resultados(dic_goles_hechos, dic_goles_recibidos, dic_puntos):
82     """
83     Calcula las métricas globales a partir de los diccionarios con estadísticas.
84
85     Argumentos:
86     - dic_goles_hechos (dict): goles marcados por cada equipo.
87     - dic_goles_recibidos (dict): goles recibidos por cada equipo.
88     - dic_puntos (dict): puntos acumulados por cada equipo.
89
90     Return:
91     - equipo_mas_goleador (str)
92     - equipo_mas_goleado (str)
93     - clasificacion (list de tuplas) ordenada descendente
94     """
95     equipo_mas_goleador = max(dic_goles_hechos, key=dic_goles_hechos.get)
96     equipo_mas_goleado = max(dic_goles_recibidos, key=dic_goles_recibidos.get)
97     clasificacion = sorted(dic_puntos.items(), key=lambda x: x[1], reverse=True)
98     return equipo_mas_goleador, equipo_mas_goleado, clasificacion
99
100 # Muestra resultados
101 def muestra_resultados(dic_goles_hechos, dic_goles_recibidos, equipo_mas_goleador, equipo_mas_goleado, clasificacion):
102     """
103     Muestra por pantalla los resultados finales del análisis de partidos, incluyendo:
104     - Goles hechos por equipo en orden descendente
105     - Goles recibidos por equipo en orden descendente
106     - Equipo más goleador y más goleado y sus valores
107     - Clasificación global según puntos.
108
109     Argumentos:
110     - dic_goles_hechos (dict): goles marcados por cada equipo.
111     - dic_goles_recibidos (dict): goles recibidos por cada equipo.
112     - equipo_mas_goleador (str): nombre del equipo más goleador.
113     - equipo_mas_goleado (str): nombre del equipo más goleado.
114     - clasificacion (lista de tuplas): clasificación ordenada por puntos.
115     Return: None (solo imprime)
116     """
117
118     print("Goles hechos por equipo:")
119     for equipo, goles in sorted(dic_goles_hechos.items(), key= lambda x: x[1], reverse = True):
120         print(f'{equipo}: {goles}')
121
122     print("\nGoles recibidos por equipo:")
123     for equipo, goles in sorted(dic_goles_recibidos.items(), key = lambda x: x[1], reverse = True):
124         print(f'{equipo}: {goles}')
125
126     print(f'\nEquipo más goleador: {equipo_mas_goleador} ({dic_goles_hechos[equipo_mas_goleador]} goles hechos)')
127     print(f'Equipo más goleado: {equipo_mas_goleado} ({dic_goles_recibidos[equipo_mas_goleado]} goles recibidos)')
128
129     print("\nClasificación global (puntos):")
130     for equipo, puntos in clasificacion:
131         print(f'{equipo}: {puntos}')
132
133 def main_partidos(enlace_archivo):
134     """
135     Función principal del programa:
136     - Lee el archivo de partidos.
137     - Inicializa los diccionarios de estadísticas.
138     - Procesa cada partido del fichero.
139     - Calcula las estadísticas.
140     - Muestra los resultados por pantalla.
141     - Retorna las estadísticas
142     Argumento: enlace a archivo que hay que procesar
143     Return:
```

```

143     Return:
144     - dic_goles_hechos (dict): goles marcados por cada equipo.
145     - dic_goles_recibidos (dict): goles recibidos por cada equipo.
146     - equipo_mas_goleador (str): nombre del equipo más goleador.
147     - equipo_mas_goleado (str): nombre del equipo más goleado.
148     - clasificacion (lista de tuplas): clasificación ordenada por puntos.
149     """
150     lineas_partido = abrir_pasar_file_lista_lineas(enlace_archivo)
151     dic_goles_hechos, dic_goles_recibidos, dic_puntos = crear_diccionarios()
152     for linea in lineas_partido:
153         procesar_partido(linea, dic_goles_hechos, dic_goles_recibidos, dic_puntos)
154     equipo_mas_goleador, equipo_mas_goleado, clasificacion = calcular_metricas_resultados(dic_goles_hechos, dic_goles_
155     muestra_resultados(dic_goles_hechos, dic_goles_recibidos, equipo_mas_goleador, equipo_mas_goleado, clasificacion)
156     return dic_goles_recibidos, dic_goles_hechos, clasificacion, equipo_mas_goleado, equipo_mas_goleador
157
158 main_partidos(r"C:\Users\vanes\OneDrive\Documenten\CURSOS\IT Academy - Anàlisis de dades\_ESPECIALITAT\Sprint 9 Python"

```

Python

Goles hechos por equipo:

Figueres: 161
 Llagostera: 159
 Vilafranca: 157
 Terrassa: 147
 Cornellà: 147

...

Goles recibidos por equipo:

Vilafranca: 172
 Terrassa: 164
 Figueres: 154
 Nàstic de Tarragona: 148
 Cornellà: 146
 Olot: 144
 RCD Espanyol: 144
 Llagostera: 142
 Sant Andreu: 134
 Cerdanyola: 133
 Lleida Esportiu: 133

...

Equipo más goleador: Figueres (161 goles hechos)
 Equipo más goleado: Vilafranca (172 goles recibidos)

Classificació global (puntos):

Girona FC: 96
 Llagostera: 94
 Sabadell: 85

Para este programa:

- creamos 5 funciones secundarias + 1 principal:
 1. abrir_pasar_file_lista_lineas(link_carpeta)
 2. crear_diccionarios()
 3. procesar_partido(linea, dic_goles_hechos, dic_goles_recibidos, dic_puntos)
 4. calcular_metricas_resultados(dic_goles_hechos, dic_goles_recibidos, dic_puntos)
 5. muestra_resultados(.....)
 6. main_partidos(enlace_archivo)

Importamos 1 librería: “re” es un módulo estándar de Python usada para trabajar con expresiones regulares. Permite:

- Buscar patrones en cadenas de texto
- Separar textos con separadores complejos
- Validar formatos (emails, números, etc.)

- Sustituir partes de un texto. En nuestro caso la usamos para separar correctamente los datos de cada partido, cuando estén en formato “cadena de texto”, y que puede tener 1 varias tabulaciones entre datos.

Función 1: abrir_pasar_file_lista_lineas(link_carpetas):

- Abre un fichero de texto en modo lectura (r: read), convierte el contenido a una lista de strings y cierra automáticamente el archivo. También avisa cuando el fichero no se encuentra (contempla Error FileNotFoundError).
- Usa un **try / except** para controlar qué pasa en una situación excepto en otra situación concreta. En este caso, cuando no se encuentra el archivo y salta **FileNotFoundException**, se muestra un mensaje al usuario avisando de ello y además “e” (alias asignado al error), para que muestre el mensaje + los detalles del error. Como hay “return” dentro de Except, el programa termina.
- **open(rutaArchivo, modo[opc]):** abre el archivo y lo devuelve como objeto. El “modo” es opcional, r = read (abre en modo lectura). También existen otros modos: w (write = modo lectura, y crea archivo si no existe); a (append = abre para añadir y crea el archivo si no existe); x (crea el archivo y da error si ya existe).
- Al añadir “**with**” al “**open()**” se automatiza el cierre del archivo. En caso contrario debería hacerse de forma manual con nombre **VariableArchivo.close()** al terminar la función.
- Cuando se llame a la función se pondrá r’ delante de la ruta para modificar automáticamente la contrabarra a la derecha y no tener que cambiarla a mano.

Podríamos aprovechar la función del ejercicio 2.1 que lee el archivo con .read() (subir_file_pasar_txt_str(link_carpetas), y luego aplicar una separación del str por líneas con .split(), pero como en este caso cada línea es el objeto de interés, así es más directo.

Función 2: crear_diccionarios()

- Crea tres diccionarios vacíos que servirán para acumular las estadísticas:
 1. dic_goles_hechos: goles marcados por cada equipo.
 2. dic_goles_recibidos: goles encajados por cada equipo.
 3. dic_puntos: puntos obtenidos por cada equipo.
- Devuelve los tres diccionarios en una tupla, que posteriormente se desempaquetará al llamar la función.

Función 3: procesar_partido(línea, dic_goles_hechos, dic_goles_recibidos, dic_puntuacion)

- Procesa una sola línea del fichero, que representa un partido de fútbol.
- Se eliminan espacios y saltos de línea al principio y al final del string con el método “`.strip()`” . Al guardar el resultado con el mismo nombre, se sobrescribe la variable string.
- Con la estructura de control “`if`” , si la línea está vacía la función termina con `return`. Así se ignoran líneas vacías sin dar error en el proceso.
- Con `x.strip() for x in re.split("\t+", linea)`, se realiza una función rápida para volver a eliminar espacios delante y detrás de los 3 elementos extraídos con “`re.split()`” , que al especificar “`\t+`” permite la separación por 1 o más tabulaciones alrededor del texto.
- El resultado es una lista de cadenas porque añadimos `[]` a los elementos extraídos.
- Si la línea no contiene exactamente 3 elementos (nombre equipo local, resultado goles y equipo visitante), se ignora y termina el proceso de esa línea con `return`.
- Ahora se desempaquetá la lista con los 3 elementos de información del partido y se asignan nombres a las variables, aprovechando el orden subyacente en cada partido: `nom_equipo_local, resultado_goles, nom_equipo_visitante`
- `resultado_goles` todavía no tiene la información separada (“`2-1` ”):
 1. Se engloba dentro de un `try / except` para evitar que el programa falle si el resultado no puede convertirse a número.
 2. con “`.split("-")`” se separa el resultado por el guión medio para obtener los goles de cada equipo.
 3. `map(funcion, iterables)` permite ejecutar la función indicada en el paréntesis para cada iterable y evita un bucle `for` para cada elemento. `Int` = convierte a entero cada resultado del partido.
 4. Se guarda bajo el nombre de 2 variables: `goles_local, goles_visitante`
- Con los datos de los goles, actualizamos los diccionarios de entrada siguiendo la estructura:
 1. `nombre_diccionario[nombre_clave] = valor_anterior + valor_adicional`
Donde `valor_anterior = nombre_diccionario.get(nombre_clave, valor por defecto si clave no existe)`
- Con el método de diccionarios “`.get(Keyname, value)`” se retorna el valor de un ítem con la llave especificada. Si se añade un valor al paréntesis (opcional), se indica el valor a retornar si la llave no existe.
Si no añadimos valor por defecto al “`.get(key, value)`” , antes de iniciar el acumulador será necesario añadir una estructura que declare un valor inicial si la llave no existe, , como:

```
if nom_equipo_local not in dic_puntos:  
    dic_puntos[nom_equipo_local] = 0
```

La función “procesar_partido(...)” **no tiene return**, ya que se modifican directamente los diccionarios entrados como argumentos.

Función 4: calcular_metricas_resultados(dic_goles_hechos, dic_goles_recibidos, dic_puntos)

- Calcula las métricas finales solicitadas:
 - o Equipo con más goles hechos: max(diccionario, key=diccionario.get)
 - o Equipo con más goles recibidos: max(diccionario, key=diccionario.get)
 - o Clasificación ordenada por puntos de mayor a menor: sorted(diccionario.items(), key=lambda x: x[1], reverse=True)
- La función incorporada (built in function) **max()** aplicada a un diccionario sólo compararía las claves del diccionario (nombres del equipo). Para comparar por los valores del diccionario (goles) y obtener el nombre del equipo (clave), hay que indicar a Python que la comparación sea por el valor asociado de la clave mediante “**key = nom_diccionario.get**” .

CLASIFICACIÓN: sorted(diccionario.items(), key=lambda x: x[1], reverse=True)

- sorted(..., key=...) ordena la lista donde key es la función que aplica a cada elemento para determinar el orden
- **dic_puntos.items()** transforma el diccionario en una colección de tuplas (equipo, puntos).
- **sorted()** ordena esas tuplas. Dentro del paréntesis, se especificará cómo ordenar: **key=lambda x: x[1]** indica que la ordenación se hace por el segundo elemento de la tupla, es decir, los puntos. Para cada tupla (equipo, puntos) usa la posición 1 (puntos) y compara.
“**lambda**” **x: ...** permite en Python crear una función anónima rápida.
“**reverse= True**” significa de mayor a menor. Por defecto es de menor a mayor (o en orden alfabético, en el caso de strings).

Con el return final devuelve los tres resultados en una tupla, que se desempaquetá al llamar la función.

Función 5: muestra_resultados(...)

- Muestra diferentes elementos por pantalla.
- Se usan bucles for y el método **.items()** para recorrer los diccionarios y obtener los datos. Imprime:
 - o Goles hechos por equipo.
 - o Goles recibidos por equipo.

- Equipo más goleador.
- Equipo más goleado.
- Clasificación final por puntos.

Función 6: llamada_funciones_doc_partidos(enlace_archivo)

- Centraliza la ejecución del programa.
1. Llama a la función de lectura del fichero cuya ruta entra como argumento y obtiene la lista de líneas.
 2. Crea los diccionarios vacíos.
 3. Recorre cada línea del fichero y llama a procesar_partido() para acumular estadísticas.
 4. Calcula las métricas finales.
 5. Muestra los resultados por pantalla.
 6. Retorna las estadísticas para su posterior uso.