NIVEL 1
EJERCICIO 1

Calculadora del índice de masa corporal

Escriba una función que calcule el IMC del usuario, es decir, quien la ejecute tendrá que introducir estos datos.
La función debe clasificar el resultado en sus respectivas categorías.

Descripción general.
Se genera un código que solicita los valores de peso y altura al usuario y en base a estos datos calcula el IMC.

Partes del código.
El programa se divide en varias funciones:

1. **datos.** La función “datos” solicita los valores de peso y altura al usuario, verifica que los datos sean correctos y gestiona las excepciones para casos de entradas no válidas.
2. **Imc.** La función “imc” calcula el valor de IMC en base del peso y altura.
3. **imc_tipo.** La función “imc_tipo” determina el tipo de IMC en base al valor de IMC.
4. **calculo_imc.** La función “calculo_imc” es una función de carácter general que ejecuta las funciones anteriores para realizar el cálculo de IMC.

Función datos.
La función “datos” puede dividirse en las siguientes partes:

1. **Entrada de datos.**
   - El código solicita la introducción de los valores de peso en kg y de altura en metros.
   - Los valores se convierten al tipo float para poder operar con valores decimales.

```
peso = float(input("Introduzca su peso en kg"))
altura =  float(input("introduzca su altura en metros"))
```

2. **Verificación de los datos.**
   - Se comprueba que el valor introducido sea un número. Si el valor no es un número se imprime un mensaje de error y se solicita al usuario que vuelva a introducir los datos.
   - Se usa una estructura try – except para comprobar que el valor introducido es un número.
   - Se utiliza While True: para crear un bucle en el que el programa vuelva a solicitar la introducción de datos.

```
def datos():
while True:
try:
peso = float(input("Introduzca su peso en kg"))
if peso <= 0:
print("Un peso de cero o menor no es valido. Intentelo de nuevo")
continue

altura =  float(input("introduzca su altura en metros"))
if altura <= 0:
```

```
print("Una altura de cero o menor no es valida. Intentelo de nuevo")
continue
break
except ValueError:
print("Dato no valido. Introduzca un número")

print(f"su peso es: {peso:.1f} kg")
print(f"su altura es: {altura:.2f} m")
return(peso, altura)
```

3. **Comprobación del rango de datos.**
   - Se comprueba que los valores introducidos sean mayores que cero. Si el valor es menor o igual a cero el código imprime un mensaje de error y solicita al usuario que vuelva a introducir los datos. Si el valor es mayor que cero se rompe el bucle y el código ejecuta el siguiente paso.
     - If “valor” > 0 : comprueba si el valor es mayor que cero.
     - While True: crea un bucle en el que el programa vuelva a solicitar la introducción de datos.
     - Continue: salta el resto del código y vuelve al principio del bucle.
     - Break: rompe el bucle y el programa continua con el siguiente código fuera del bucle.

```
def datos():
while True:
try:
peso = float(input("Introduzca su peso en kg"))
if peso <= 0:
print("Un peso de cero o menor no es valido. Intentelo de nuevo")
continue

altura =  float(input("introduzca su altura en metros"))
if altura <= 0:
print("Una altura de cero o menor no es valida. Intentelo de nuevo")
continue
break
except ValueError:
print("Dato no valido. Introduzca un número")

print(f"su peso es: {peso:.1f} kg")
print(f"su altura es: {altura:.2f} m")
return(peso, altura)
```

4. **Resultados.** El programa devuelve e imprime los valores de peso y altura

```
  print(f"su peso es: {peso:.1f} kg")
  print(f"su altura es: {altura:.2f} m")
 return(peso, altura)
```

Función imc(peso, altura)

- Calcula el valor_imc en función del peso y la altura.
- Devuelve e imprime el valor_imc.
  - :1f. Imprime el resultado con un decimal de precisión.

```
def imc(peso, altura):
    valor_imc = peso / (altura ** 2)

    print(f"su imc es: {valor_imc:.1f}")
    return(valor_imc)
```

Función imc_tipo(valor_imc)

- Clasifica el tipo_imc en función del valor_imc.
- Devuelve e imprime el tipo_imc.

```
def imc_tipo(valor_imc):
    if valor_imc < 18.5:
        tipo_imc = "Peso Bajo"
    elif valor_imc < 24.9:
        tipo_imc = "Peso Normal"
    elif valor_imc < 29.9:
        tipo_imc = "Sobrepeso"
    else:
        tipo_imc = "Obesidad"

    print(f"su tipo_imc es: {tipo_imc}")
```

Función calculo_imc()

La función calculo_imc es una función general que integra todas las demás:

- Llama a la función datos() para obtener el peso y la altura del usuario.
- Llama a la función imc() para calcular el IMC con los datos proporcionados.
- Llama a la función imc_tipo() para clasificar el IMC calculado.

```
def calculo_imc():
    peso, altura = datos()
    valor_imc = imc(peso, altura)
    imc_tipo(valor_imc)
```

CÓDIGO COMPLETO

```python
def datos():
    while True:
        try:
            peso = float(input("Introduzca su peso en kg"))
            if peso <= 0:
                print("Un peso de cero o menor no es valido. Intentelo de nuevo")
                continue
            altura =  float(input("introduzca su altura en metros"))
            if altura <= 0:
                print("Una altura de cero o menor no es valida. Intentelo de nuevo")
                continue
            break
        except ValueError:
            print("Dato no valido. Introduzca un número")

    print(f"su peso es: {peso:.1f} kg")
    print(f"su altura es: {altura:.2f} m")
    return(peso,altura)

def imc(peso,altura):
    valor_imc = peso / (altura ** 2)

    print(f"su imc es: {valor_imc:.1f}")
    return(valor_imc)

def imc_tipo(valor_imc):
    if valor_imc < 18.5:
        tipo_imc = "Peso Bajo"
    elif valor_imc < 24.9:
        tipo_imc = "Peso Normal"
    elif valor_imc < 29.9:
        tipo_imc = "Sobrepeso"
    else:
        tipo_imc = "Obesidad"

    print(f"su tipo_imc es: {tipo_imc}")

def calculo_imc():
    peso,altura = datos()
    valor_imc = imc(peso,altura)
    imc_tipo(valor_imc)
calculo_imc()
```

✓ 7.7s Python

```
su peso es: 75.0 kg
su altura es: 1.80 m
su imc es: 23.1
su tipo_imc es: Peso Normal
```

NIVEL 1
EJERCICIO 2

Convertidor de temperaturas.

Hay varias unidades de temperatura utilizadas en diferentes contextos y regiones. Los más comunes son Celsius (C), Fahrenheit (F) y Kelvin (K). También hay otras unidades como Rankine y Reaumur. Seleccione al menos 2 conversores, de modo que al entrar en una temperatura, devuelva al menos dos conversiones.

Descripción general:

Se genera un código que permite al usuario introducir una temperatura en grados Celsius y luego convierte esa temperatura a otras escalas de temperatura: Fahrenheit, Kelvin, Rankine y Reaumur. Finalmente, el programa muestra los resultados de estas conversiones.

El código realiza los siguientes pasos:

1. **Entrada de datos:** El programa solicita al usuario que introduzca una temperatura.
2. **validación de datos**: El programa valida que el valor introducido sea numérico y válido.
3. **Cálculos de las conversiones de temperatura**: Calcula el valor de la temperatura en las otras escalas.
4. **Impresión de resultados**: Muestra las conversiones.

Partes del código:

El código puede dividirse en las siguientes partes:

1. **Entrada y validación de los datos:**
   - El programa solicita al usuario que introduzca una temperatura en grados Celsius mediante la función input().
   - El valor ingresado se convierte a tipo float para que pueda ser tratado como un número decimal.
   - Se realiza una comprobación de los datos introducidos. Si el dato es erróneo o esta fuera del rango establecido el programa muestra un error e indica al usuario que vuelva a introducir el dato.

```
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Inténtelo de nuevo")
        else:
            break
    except ValueError:
        print("Datos erróneos. Inténtelo de nuevo")
```

- **Entrada de datos:**
  - El programa solicita al usuario que introduzca una temperatura.

```
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Inténtelo de nuevo")
        else:
            break
    except ValueError:
        print("Datos erróneos. Inténtelo de nuevo")
```

- **Validación de la temperatura**:
  - Si el valor ingresado es menor o igual a -273.15 °C, el programa informa al usuario que la temperatura no es válida, ya que el cero absoluto (la temperatura más baja posible) es -273.15°C. Luego le pide al usuario que intente de nuevo.

```
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Inténtelo de nuevo")
        else:
            break
    except ValueError:
        print("Datos erróneos. Inténtelo de nuevo")
```

- **Manejo de errores**:
  - Si el valor ingresado no puede ser convertido a un número flotante (por ejemplo, si el usuario ingresa texto en lugar de un número), se maneja el error con un bloque except ValueError que imprime un mensaje pidiendo al usuario que ingrese un valor válido.

```
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Inténtelo de nuevo")
        else:
            break
    except ValueError:
        print("Datos erróneos. Inténtelo de nuevo")
```

- **Continuación o finalización**:
  - El programa sigue solicitando entradas hasta que el usuario ingresa un valor válido y mayor a -273.15°C, momento en el cual el bucle while se rompe mediante el comando break.

```
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Inténtelo de nuevo")
        else:
            break
    except ValueError:
        print("Datos erróneos. Inténtelo de nuevo")
```

2. **Cálculos de las conversiones de temperatura:**
   - Una vez que la entrada ha sido validada, se crean las distintas variables para la conversión de la temperatura Celsius a las otras escalas de temperatura.

```
fahrenheit = (celsius * 9/5) + 32
kelvin = celsius + 273.15
rankine = (celsius * 9/5) + 491.67
reaumur = celsius * 4/5
```

3. **Impresión de los resultados:**
   - Una vez realizadas las conversiones, el programa imprime los resultados.

```
print(f"{celsius:.1f} grados Celsius equivalen a:")
print(f"{fahrenheit:9.1f} grados Fahrenheit")
print(f"{kelvin:9.1f} grados Kelvin")
print(f"{rankine:9.1f} grados Rankine")
print(f"{reaumur:9.1f} grados Reaumur")
```

     - **Formato de salida**:
       - **f””** Permite combinar variables ({}) con texto en la impresión.
       - **:9** Hace que cada valor se imprima con un ancho de 9 caracteres, alineado a la derecha.
       - **:.1f** Hace que cada valor se imprima con solo un decimal.

CÓDIGO COMPLETO

```python
while True:
    try:
        celsius = float(input("introduce una temperatura en grados Celsius"))
        if celsius <= -273.15:
            print("la temperatura debe ser superior a -273.15 grados celsius")
            print("Intentelo de nuevo")
        else:
            break

    except ValueError:
        print("Datos erroneos. Intentelo de nuevo")

fahrenheit = (celsius * 9/5) + 32
kelvin = celsius + 273.15
rankine = (celsius * 9/5) + 491.67
reaumur = celsius * 4/5

print(f"{celsius:.1f} grados Celsius equivalen a:")
print(f"{fahrenheit:9.1f} grados Fahrenheit")
print(f"{kelvin:9.1f} grados Kelvin")
print(f"{rankine:9.1f} grados Rankine")
print(f"{reaumur:9.1f} grados Reaumur")
```

✓ 0.0s Python

```
35.0 grados Celsius equivalen a:
     95.0 grados Fahrenheit
    308.1 grados Kelvin
    554.7 grados Rankine
     28.0 grados Reaumur
```

NIVEL 1
EJERCICIO 3

Contador de las palabras de un texto.
Escribe una función que dado un texto, muestra las veces que aparece cada palabra.

Descripción general:
Se genera un código que crea una función llamada contador_palabras, que dado un texto como entrada, muestra la frecuencia de aparición de cada palabra en el texto.

El código hace uso de los siguientes módulos:
1. **re**: Para trabajar con expresiones regulares y realizar manipulaciones de texto (en este caso, eliminar caracteres no deseados).
2. **collections.Counter**: Para contar las ocurrencias de las palabras de manera eficiente.

La función realiza los siguientes pasos:
1. **Formateo del texto**: Eliminación de los caracteres no alfanuméricos en el texto.
2. **Lista de palabras**: Creación de una lista de palabras en minusculas a partir del texto.
3. **Contar las palabras:** Se cuenta el número de veces que aparece cada palabra en el texto.
4. **Mostrar los resultados:** Se imprimen las palabras y el número de veces que aparecen en el texto.

Partes del código:
El código puede dividirse en la siguientes partes:

1. **Importación de módulos:**
   - Se importan los módulos re y collections.Counter.
     - **re**: Es el módulo de expresiones regulares de Python, que permite realizar búsquedas y manipulaciones de texto con patrones.
     - **Counter**: Es una clase de collections que ayuda a contar las ocurrencias de elementos en un iterable, como una lista o una cadena.

```
import re
from collections import Counter
```

2. **Definición de la función** contador_palabras**:**
   - La función contador_palabras toma un único argumento texto, que es una cadena que representa el texto sobre el que se desea realizar el análisis de palabras.

```
def contador_palabras(texto):
```

3. **Formato del texto (eliminación de caracteres no alfanuméricos):**
   - Se utiliza el módulo re para eliminar caracteres no alfanuméricos y de puntuación del texto.
     - **[^\w\s]** Selecciona cualquier carácter que **no** sea una palabra alfanumérica (\w que incluye letras, números y guion bajo) ni un espacio en blanco (\s).

- **re.sub()** Reemplaza todas las coincidencias de este patrón con una cadena vacía (''), eliminando esos caracteres del texto.

```
texto_limpio = re.sub(r'[^\w\s]', '', texto)
```

4. **Transformación del texto en minúsculas y separación en palabras:**
   - Crea una lista en minúsculas de las palabras en el texto formateado.
     - **.lower()** Convierte el texto formateado a minúsculas.
     - **.split()** Divide el texto en una lista de palabras. Por defecto las palabras son separadas por espacios.

```
palabras_texto = texto_limpio.lower().split()
```

5. **Contar las ocurrencias de cada palabra:**
   - Counter(palabras_texto):
     - La clase Counter de collections cuenta la frecuencia de cada palabra en la lista.
     - El resultado es un objeto Counter que es un diccionario, donde las claves son las palabras y los valores son el número de veces que cada palabra aparece en el texto.

```
resultado = Counter(palabras_texto)
```

6. **Imprimir los resultados ordenados alfabeticamente por palabra:**
   - sorted(resultado.items()):
     - resultado.items() devuelve una lista de tuplas (clave, valor) donde cada tupla contiene una palabra del texto y su frecuencia.
     - sorted() ordena estas tuplas alfabéticamente según la clave (la palabra).

   - print(f"{k}: {v}"):
     - Se imprime cada palabra k seguida de su frecuencia v.
     - El formato f"{k}: {v}" es una cadena formateada que imprime las palabras y sus conteos de manera legible.

```
for k,v in sorted(resultado.items()):
   print(f"{k}: {v}")
```

CODIGO COMPLETO

```python
import re
from collections import Counter

def contador_palabras(texto):

    texto_limpio = re.sub(r'[^\w\s]', '', texto)
    palabras_texto = texto_limpio.lower().split()

    resultado = Counter(palabras_texto)

    for k,v in sorted(resultado.items()):
        print(f"{k}: {v}")
```

✓ 0.0s Python

EJEMPLO DE USO

```python
texto = " En un lugar de la mancha de cuyo nombre..."
contador_palabras(texto)
```

✓ 0.0s Python

```
cuyo: 1
de: 2
deco: 1
dedo: 1
en: 1
la: 1
lugar: 1
mancha: 1
nombre: 1
un: 1
```

NIVEL 1
EJERCICIO 4

Diccionario inverso.
Resulta que el cliente tiene una encuesta muy antigua que se almacena en un diccionario y los resultados necesitan ser invertidos, es decir, se intercambian las claves y los valores. Los valores y las claves del diccionario original son únicos; si no es así, la función debe ser impresa en un mensaje de advertencia.

Descripción del Código:
Se crea un programa que crea un diccionario inverso a uno dado si en el diccionario original no existen valores repetidos.

El programa realiza los siguientes pasos:

1. **Comprobación de valores repetidos**. Se comprueba si el diccionario original tiene valores repetidos.
2. **Creación de un diccionario invertido**. Se crea un diccionario invertido si no hay valores repetidos.
3. **Visualización de errores**. Se muestra un mensaje de error si hay valores repetidos.

Partes del código:
El código puede dividirse en las siguientes partes:

1. **Comprobación de valores repetidos**.
   - Se crea una lista con los valores del diccionario.

```
lista_valores = list(dictionary.values())
```

   - Se crea una lista con los valores únicos del diccionario.

```
lista_valores_sin_duplicados = list(dict.fromkeys(lista_valores))
```

   - Se comparan ambas listas.

```
if lista_valores == lista_valores_sin_duplicados:
```

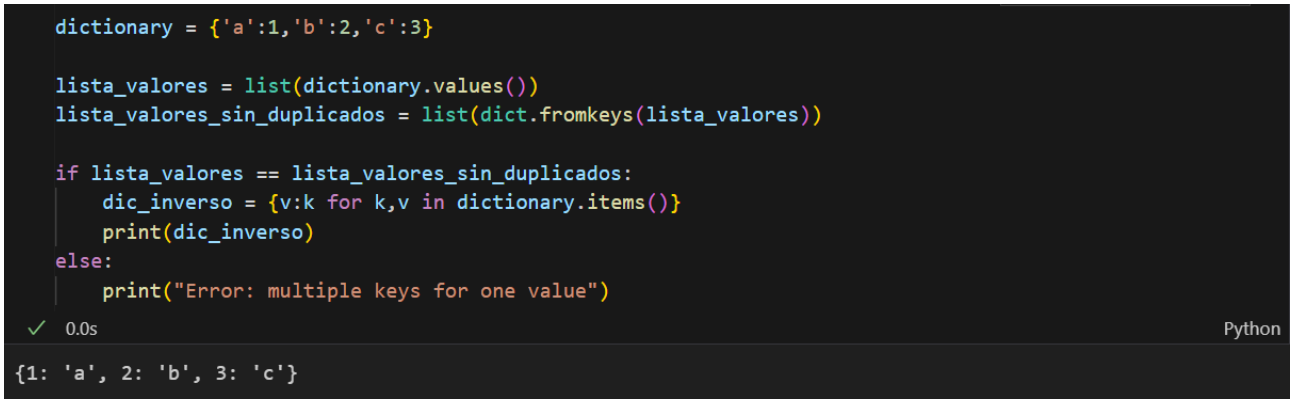
2. **Creación del diccionario invertido.**
   - Si no hay valores repetidos, se crea un nuevo diccionario con los elementos invertidos del diccionario original. Es decir, los valores se convierten en claves y las claves en valores.
   - Impresión del diccionario inverso.

```
dictionary = {'a':1,'b':2,'c':3}
lista_valores = list(dictionary.values())
lista_valores_sin_duplicados = list(dict.fromkeys(lista_valores))
if lista_valores == lista_valores_sin_duplicados:
    dic_inverso = {v:k for k,v in dictionary.items()}
    print(dic_inverso)
else:
    print("Error: multiple keys for one value")
```

3. **Visualización de error.** Si hay valores repetidos, se imprime un mensaje de error.

```
dictionary = {'a':1,'b':2,'c':3}
lista_valores = list(dictionary.values())
lista_valores_sin_duplicados = list(dict.fromkeys(lista_valores))
if lista_valores == lista_valores_sin_duplicados:
    dic_inverso = {v:k for k,v in dictionary.items()}
    print(dic_inverso)
else:
    print("Error: multiple keys for one value")
```

CODIGO COMPLETO



NIVEL 2
EJERCICIO 1

Diccionario inverso con duplicados

Continuando con el ejercicio 4 del nivel 1: el cliente se ha olvidado de comentar un detalle y resulta que los valores en el diccionario original se pueden duplicar y, lo que es más importante, las claves se pueden duplicar. En este caso, en el ejercicio anterior, se imprimirá un mensaje de advertencia, ahora los valores del diccionario resultante tendrán que ser almacenados como lista. Tenga en cuenta que si se trata de un valor único, no debe ser una lista.

Descripción del Código:

Se genera un código que dado un diccionario crea un diccionario invertido del original, es decir, que los valores se convierten en claves y las claves en valores.
Si los valores están repetido en el diccionario original, se tomará el valor como clave y las claves originales se almacenarán como valores en forma de listas en el nuevo diccionario.

El código hace uso de los siguientes módulos:

- **collections.Counter**. Crea un diccionario con los elementos de un iterable como clave y el número de veces que aparece ese elemento como valor.
- **collections.defaultdict.** Crea un diccionario con valores iniciales.

Partes del Código.

El código puede dividirse en las siguientes partes:

- **Importación de los módulos.**
  Se importan los módulos collections.Counter y collections.defaultdict.

  ```
  from collections import Counter, defaultdict
  ```

- **Comprobación de las repeticiones.**
  Se crea un diccionario con el número de veces que aparece un valor en el diccionario original.

  ```
  dic_veces_valor = Counter(dic.values())
  ```

- **Creación del diccionario de resultado con valores iniciales.**
  Se crea un diccionario con valores iniciales.

  ```
  reverse_dictionary_plus = defaultdict(list)
  ```

- **Calculo de los valores del diccionario.**
  Se crea un par de diccionario con el valor del diccionario original como clave y la clave original como valor, si el valor del diccionario original solo aparece una vez.
  Si el valor del diccionario original se repitiese, en el diccionario de resultado se situaría el valor como clave y una lista de las claves originales como valor.

```
        for k,v in dic.items():
    if dic_veces_valor[v] == 1:
        reverse_dictionary_plus[v] = k
    else:
        reverse_dictionary_plus[v].append(k)
```

- **Impresión de resultados.**
  Se imprime el diccionario de resultado como un diccionario estándar.

CÓDIGO COMPLETO.

```python
from collections import Counter, defaultdict

dic = {'x':'apple','y':'banana','z':'banana'}

dic_veces_valor = Counter(dic.values())
reverse_dictionary_plus = defaultdict(list)

for k,v in dic.items():
    if dic_veces_valor[v] == 1:
        reverse_dictionary_plus[v] = k
    else:
        reverse_dictionary_plus[v].append(k)

print(dict(reverse_dictionary_plus))
```

✓ 0.0s Python

```
{'apple': 'x', 'banana': ['y', 'z']}
```

NIVEL 2
EJERCICIO 2

Conversión del tipo de datos
El cliente recibe una lista de datos y necesita generar dos listas, la primera en la que todos los elementos podrían convertirse en flotantes y el otro donde los elementos no pudieron ser convertidos.

Descripción General.
Se crea un código que separa los elementos de una lista, tanto los individuales como los que están integrados en una tupla o una sublista.

Partes del código.
El código se puede dividir en las siguientes partes:

- **Creación de las listas de resultados.** Se crean dos listas para almacenar los elementos según puedan ser del tipo flotante o no.

```
flotantes = []
no_flotantes = []
```

- **Almacenamiento de los resultados.** Se almacenan los datos en las listas correspondientes en función de si los datos pueden ser convertidos a tipo flotante o no.
  - Si los elementos están en una tupla o sublista, se comprueba si pueden ser convertidos a tipo float o no y se almacenan en la lista correspondiente.
  - Si los elementos no están en una tupla o sublista, se comprueba si pueden ser convertidos a tipo float o no y se almacenan en la lista correspondiente.

```
for n in conversion:
    if isinstance(n,(list,tuple)):
        for item in n:
            try:
                flotantes.append(float(item))
            except ValueError:
                no_flotantes.append(item)
    else:
        try:
            flotantes.append(float(n))
        except ValueError:
            no_flotantes.append(n)
```

- **Impresión de los resultados.** Se imprimen las dos listas con los elementos correspondientes.

```
print(flotantes)
print(no_flotantes)
```

CÓDIGO COMPLETO.

```python
conversion = ['1.3', 'one', '1e10', 'seven', '3-1/2', ('2',1,1.4,'not-a-number'),[1,2,'3','3.4']]

flotantes = []
no_flotantes = []

for n in conversion:
    if isinstance(n,(list,tuple)):
        for item in n:
            try:
                flotantes.append(float(item))
            except ValueError:
                no_flotantes.append(item)

    else:
        try:
            flotantes.append(float(n))
        except ValueError:
            no_flotantes.append(n)

print(flotantes)
print(no_flotantes)
```

✓ 0.0s Python

```
[1.3, 10000000000.0, 2.0, 1.0, 1.4, 1.0, 2.0, 3.0, 3.4]
['one', 'seven', '3-1/2', 'not-a-number']
```

NIVEL 3
EJERCICIO 1

Contador y editor de palabras de un texto.

El cliente estaba contento con el contador de palabras, pero ahora quiere leer los archivos TXT y calcular la frecuencia de cada palabra en las letras habituales del diccionario, según la letra, es decir, las claves tienen que ir de la A a la Z y dentro de la A de la A a la Z.

Descripción General.

Se genera un código que muestra las repeticiones de cada palabra en un archivo txt. La salida se organiza por la letra inicial alfabéticamente y las palabras dentro de cada grupo también se organizan alfabéticamente.

El código hace uso de los módulos string, collections.Counter y collections.defaultdict.

- **string**. El módulo string proporciona funciones para trabajar con cadenas de texto.
- **collections.Counter**. Proporciona un método eficaz para contar los elementos de un iterables.
- **collections.defaultdict**. Crea un diccionario con valores iniciales.

Partes del código.

El código puede dividirse en las siguientes partes:

- **Lectura del archivo .txt** . Se abre el archivo txt y se procede a su lectura.
-

```
with open("tu_me_quieres_blanca.txt", "r") as archivo:
        texto = archivo.read()
```

- **Formateo del texto y creación de la lista de palabras.** Se eliminan los signos de puntuación del texto y se crea un lista de palabras del texto. Se ordena la lista de palabras.

```
translator = str.maketrans('', '', string.punctuation)
lista_palabras = texto.translate(translator).lower().split()
lista_palabras.sort()
```

- **Calculo de las repeticiones.** Se crea un diccionario con las palabras del texto como clave y el número de repeticiones como valor.

```
dic_num_palabras = Counter(lista_palabras)
```

- **Creación del diccionario de resultados.** Se crea un diccionario con la inicial de la palabra como clave y un diccionario anidado como valor. El diccionario anidado tendrá la palabra como clave y el número de repeticiones como valor.
  - **creación del diccionario inicial.** Se crea un diccionario con un diccionario anidado como valor.

```
dic_resultado = defaultdict(dict)
```

  - **inserción de valores en el diccionario**. Se insertan los valores correspondientes en el diccionario.

```
for palabra, num in dic_num_palabras.items():
    dic_resultado[palabra[0]][palabra] = num
```

- **impresión de resultados**. Se imprimen los valores del diccionario de resultado.
  - Primero, se imprime la letra inicial.
  - Segundo, se imprimen las palabras y las repeticiones de cada palabra. Estos datos se imprimen con una sangría de 3 con respecto a la letra inicial para una mejor visualización.

```
for letra, palabras in dic_resultado.items():
    print(f"{letra}:")
    for palabra,num in palabras.items():
        print(f"{' '*3}{palabra}: {num}")
    print()
```

CÓDIGO COMPLETO.

```
import string
from collections import Counter,defaultdict

with open("tu_me_quieres_blanca.txt", "r") as archivo:
    texto = archivo.read()

translator = str.maketrans('', '', string.punctuation)
lista_palabras = texto.translate(translator).lower().split()
lista_palabras.sort()

dic_num_palabras = Counter(lista_palabras)
dic_resultado = defaultdict(dict)

for palabra, num in dic_num_palabras.items():
    dic_resultado[palabra[0]][palabra] = num

for letra, palabras in dic_resultado.items():
    print(f"{letra}:")
    for palabra,num in palabras.items():
        print(f"{' '*3}{palabra}: {num}")
    print()
```

✓ 0.0s Python

```
a:
   a: 3
   agua: 1
   al: 2
   alba: 4
   alcobas: 1
   alimenta: 1
   alma: 1
   amarga: 1
   azucena: 1

b:
   baco: 1
```