Objetivo

Utilizar programación modular, diseñando y usando funciones en programas con Python que contengan datos estructurados inmutables – ***strings*** y mutables – **listas**, incluyendo listas de listas.

Ejercicios con strings (cadenas de caracteres)

1. **Identificador válido**. Para que los identificadores o nombres de las variables o funciones sean válidos deben comenzar e incluir letras a..z. (minúscula o mayúscula) o el guión bajo (_). Pueden incluir también números (0..9) pero que no sean el primer carácter del identificador. Diseña una función `FirstChar(s)` que, dado un string `s` (no vacío), nos devuelva True o False si el string es válido o no para identificar o dar nombre a una variable o función[1].

```
>>> FirstChar('paciente001')
True
>>> FirstChar('P001')
True
>>> FirstChar('1Pac')
False
>>> FirstChar('_001')
True
>>> FirstChar(':p001')
False
```

2. **Porcentaje de vocales**. Escribe una función `porcentVocal(s)` en que dado un string `s`, la función devuelva el porcentaje de vocales que contiene el string. Deben considerarse vocales minúsculas y mayúsculas. Devolver el resultado con un decimal de precisión.
Consideraremos que las vocales están sin tilde o acento gráfico. Se valorará prever el caso que se envíe como argumento un string vacío.

```
>>> porcentVocal('Hola')
50.0
>>> porcentVocal('Acacia')
66.7
>>> porcentVocal('Brrrrrrr')
0.0
>>> porcentVocal('aAe')
100.0
```

3. **Nuevo string**. Diseña una función `nuevo_string (s, n)` que, dado un string `s` y un entero `n` ≥ 0, devuelva el string resultante de repetir cada vocal de `s` exactamente `n` veces en el lugar donde se encuentra situada en `s`.

```
>>> nuevo_string('Charleston', 2)
'Chaarleestoon'
```

[1] Python permite identificadores con vocales acentuadas, como `área` o caracteres alfabéticos como la `ñ`, aunque no es recomendable esta práctica por si se cambia de lenguaje de programación. Considerad para este ejercicio caracteres alfabéticos de ASCII básico (alfabeto inglés).

```
>>> nuevo_string('RDT11', 1)
'RDT11'
>>> nuevo_string('H2O', 3)
'H2OOO'
```

4. **Notas al pie de página**. Diseña una función `notas_al_pie(s)` que, dado un string `s` formado sólo por letras, signos de puntuación y asteriscos que indican una llamada a una nota al pie de página, devuelva un string donde cada asterisco es sustituido por un número entre paréntesis que indica el número de nota. El primer * se substituye por (1), el segundo por (2), etc. Ejemplos:

```
>>> notas_al_pie('Esta es la primera nota*; y esta la segunda*.')
'Esta es la primera nota(1); y esta la segunda(2).'
>>> notas_al_pie('Esta frase no tiene notas. Esta otra tampoco.')
'Esta frase no tiene notas. Esta otra tampoco.'
>>> notas_al_pie('*,*. *.')
'(1),(2). (3).'
>>> notas_al_pie('*')
'(1)'
>>> notas_al_pie('')
''
```

5. **Calcula código**. Dado un string `s` que contiene los nombres y apellidos de una persona, diseña la función `codigo(s)` que devuelva el string **ini** + str(**count**), donde **ini** contiene las iniciales de la persona (las letras mayúsculas de `s`) y **count** es el número total de letras de sus nombres y apellidos (es decir, las letras de `s` sin contar caracteres blancos o espacios). Ejemplo:

```
>>> codigo('Mireia Belmonte García')
'MBG20'
>>> codigo('Bruce Frederick Joseph Springsteen')
'BFJS31'
>>> codigo('')
''
>>> codigo('Gerard Piqué Bernabéu')
'GPB19'
>>> codigo('Sergio Ramos García')
'SRG17'
```

6. **Contador de hidrógenos**. Una fórmula química es una representación convencional de los elementos que forman un compuesto. Por ejemplo, el 1-2-butadiol sería C2H5O, que se representa con el string `'C2H5O'`. También pueden aparecer elementos químicos de dos caracteres como el calcio Ca en CaCO3 ('CaCO3') o el hierro Fe en Fe3O4 ('Fe3O4'). En estos casos el segundo carácter del símbolo siempre es una minúscula. Diseña la función `contar_hidrogenos(s)` que, dado un string **s** con un compuesto como los descritos antes, devuelve el número de **átomos de hidrógeno** que contiene. Para simplificar el problema, limitaremos el número que puede seguir el símbolo de un elemento a un valor entre **2** y **9**.

```
>>> contar_hidrogenos('HIO')
1
>>> contar_hidrogenos('H2O')
2
>>> contar_hidrogenos('C2H5O')
```

```
5
>>> contar_hidrogenos('Fe3O4')
0
>>> contar_hidrogenos('C2OH')
1
```

Ejercicios con listas

7. Diseña la función `mediaTempRang(lst)` en que, dada una lista `lst` de medidas de temperatura en ºC de un experimento, calcule y devuelva el valor medio de aquellas temperaturas de la lista que estén en el rango de 15 a 45 ºC, inclusive [15, 45]. Devolver el resultado redondeado a 2 cifras decimales. También considerar el caso en que ninguna medida de temperatura de la lista esté en el rango dado. En este caso la función devuelve el valor -1.

```
def mediaTempRang(lst):
    """
    >>> lst1 = [34.5, 12.9, 15, 43, 51.4, 23.4]
    >>> mediaTempRang(lst1)
    28.98
    >>> mediaTempRang([45.5, 12.9, 15, 32.5, 51.4, 21.2])
    22.9
    >>> mediaTempRang([14.5, 12.6, 47.8])
    -1
    >>> mediaTempRang([15, 16, 14, 50, 17])
    16.0
    """
```

8. El ***umbral*** de nivel de presión del sonido (*sound pressure level*, SPL) del oído humano es aproximadamente de 20 µP a frecuencias medias de la voz. Este valor se considera el nivel de presión de sonido (SPL) de referencia, 0 dB ($20\log_{10}(20/20)$). La función **`SPL_dB(P)`** recibe una presión acústica en µP, y devuelve su equivalente en dB.

   Diseñar la función `detect2ndNdB(lst, N)`, en que dada una lista `lst` de valores de nivel de presión de sonido (SPL) en µP, y un valor `N` (dB), busque y devuelva el valor de la segunda presión SPL que sea al menos de `N` dB. Por ejemplo, N = 30 dB es el nivel de un susurro en el oído, N = 50 dB es el nivel de una conversación normal, N = 80 dB es el nivel de ruido en una calle con tráfico. En caso de no encontrar una segunda presión SPL se ha de devolver -1.

   Notas: (i) Se debe usar la función `SPL_dB(P)` que convierte en dB la presión de entrada en µP. (ii) Se debe buscar la 2ª presión de acuerdo a la ubicación en la lista original del argumento.

   Por ejemplo, en:

```
>>> detect2ndNdB([90, 590, 750, 632, 650, 660, 2000, 789, 545], 30)
650
```

   Las presiones marcadas en rojo superan los 30 dB, pero la segunda de la lista original, 650, es la que se tiene que devolver.

```
from math import log10
def  SPL_dB(P):
    """ Calcula Sound Pressure Level en dB: 20log(x/20) x en microPascales
    """
    return 20*log10(P/20)
```

```
def detect2ndNdB(lst, N):
    """
    >>> detect2ndNdB([90,590,750,632, 650, 900, 2000, 789, 545], 30)
    650
    >>> detect2ndNdB([90,590,750,632, 650, 900, 2000, 789, 545], 33)
    2000
    >>> detect2ndNdB([90,590,750,632, 630, 600, 200, 589, 545], 30)
    -1
    >>> detect2ndNdB([9e3,1e4,1.1e5,2.2e5, 1.3e6, 2.5e6, 3.2e6], 83)
    2500000.0
    >>> detect2ndNdB([2000, 2450.5, 2500 , 456.7, 1567.8], 42)
    -1
    """
```

9. **Primos pitagóricos**. Diseña la función `primoPitagoric2(lst)` en que, dada una lista de números enteros positivos no repetidos, devuelva una lista con los 2 primeros números primos pitagóricos. Si no hubiera al menos 2 primos pitagóricos la función devuelve -1
   Un número **primo es pitagórico** si se puede escribir como la suma de dos cuadrados. Por ejemplo $5 = 2^2 + 1^1$ o $13 = 2^2 + 3^2$. Fermat demostró que un primo pitagórico p es igual a 4k + 1, para algún valor de k entero positivo. Esta condición se puede expresar como: *un número primo p es pitagórico si p modulo 4 es igual a 1, es decir si el residuo de dividir p entre 4 es 1*.

   Se debe usar la función `es_primo(n)` para evaluar si un número es primo o no.

```
def es_primo(n):
    if n <= 1: return False
    for d in range(2, n//2+1):
        if n % d == 0:
            return False
    return True

def primoPitagoric2(lst):
    """
    Ejemplos:
    >>> primoPitagoric2([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
    [5, 13]
    >>> primoPitagoric2([5, 9, 13, 17, 21, 25, 29, 33, 37, 41])
    [5, 13]
    >>> primoPitagoric2([41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81])
    [41, 53]
    >>> primoPitagoric2([3, 4, 5, 6, 7, 8, 9, 10])
    -1
    >>> lista = [81, 85, 89, 93, 97, 101, 105, 109, 113, 117, 121]
    >>> primoPitagoric2(lista)
    [89, 97]
    """
```

Ejercicios de listas de listas

10. **Contar positivos**. Dada una lista de listas que representa una matriz cuadrada `m`, diseña una función `contar_pos(m)`, que cuente los números positivos que tiene. Ejemplo:

```
>>> contar_pos([[1, -2, 3],[-4,5,6],[7,8,-9]])
6
```

11. **Mayor densidad**. Se dispone del nombre, masa y volumen de un planeta almacenado en una lista: [nombre, masa, volumen]. Además, se tiene una lista de planetas como una lista de listas de la forma: `[[nombre1, masa1, volumen1], [nombre2, masa2, volumen2], ...]`. Diseña la función `mas_denso(Lst)`, en que dada una lista de planetas `Lst`, nos devuelva el nombre del planeta más denso de esa lista. Si hubiera más de uno con la misma densidad, se devuelve el primero que encuentre en la lista original. Ejemplo:

```
>>> mas_denso([['Marte', 1, 2], ['Tierra', 2, 3], ['Venus', 1, 3]])
'Tierra'
```

12. **Fútbol**. Se dispone en una lista (equipo) de listas (jugadores) con los registros de los jugadores. En cada registro se guarda: su número de dorsal, nombre, si es comunitario o no (booleano, comunitario: True), edad y la distancia recorrida en kilómetros en los partidos jugados en el último mes.

    No todos los jugadores han jugado todos los partidos del mes, por lo que aparecerá solo la distancia recorrida de los partidos jugados.

    Diseñar una función `jugComKm(equipo, x)` en que, dada una lista de un `equipo` de futbol y un número `x` de kilómetros recorridos, nos devuelva la lista de nombres de los jugadores **comunitarios** que han recorrido de media (promedio) más de `x` km en los partidos jugados.

    De no encontrarse jugadores con este recorrido, se devolverá la lista vacía.

    Notas: pudiera haber algún jugador sin partidos jugados y en este caso el promedio lo consideramos 0.

    Se valorará devolver la lista de nombres ordenada alfabéticamente.

```
def jugComKm(lst, x):
    """
    Ejemplo:
    >>> lst_equipo = [[3, 'Pique', True, 33, 10.2, 9.0], \
                      [4, 'Ramos', True, 34, 11.0, 11.1, 9.8, 8.5], \
                      [6, 'Koke', True, 27, 7.5, 9.6, 10.3, 6.5, 5.6], \
                      [7, 'Joao',  True, 25, 10.5, 8.4, 9.0, 8.6], \
                      [8, 'Saul', True, 24, 9.5, 8.9, 10.0, 9.6], \
                      [9, 'Suarez', False, 33, 8.6, 7.5], \
                      [10, 'Lionel', False, 33, 10.0, 11.1, 9.8, 8.5,10.1], \
                      [19, 'Odriozola', True, 25, 9.5], \
                      [14, 'Araujo', False, 21, 8.9, 9.5], \
                      [15, 'Valverde', False, 22, 9.9, 10.2], \
                      [16, 'Pedri', True, 18, 10.5, 11, 9.5, 10.6], \
                      [22, 'Hermoso', False, 23, 10, 7.5, 6.6], \
                      [23, 'Iago', True, 33, 11.1, 9.0, 9.3, 8.8]]
    >>> jugComKm(lst_equipo, 10)
    ['Pedri', 'Ramos']
    >>> jugComKm(lst_equipo, 10.2)
    ['Pedri']
    >>> jugComKm(lst_equipo, 10.5)
    []
    >>> jugComKm(lst_equipo, 9.5)
    ['Iago', 'Pedri', 'Pique', 'Ramos']
    >>> jugComKm(lst_equipo, 9.4)
    ['Iago', 'Odriozola', 'Pedri', 'Pique', 'Ramos', 'Saul']
    """
```