Herramientas de Programación

Ejercicios 2.1 Datos estructurados inmutables: **strings**

1. **Máximo de un string**. Diseña una función **`máximo(str)`** que, dado un string, busque y devuelva la mayor letra minúscula (en el orden de la tabla ASCII básico). Si el string entra vacío la función debe devolver **False**. Ejemplo:

```
>>> maximo("")
False
>>> maximo("abaco")
'o'
>>> maximo("india")
'n'
>>> maximo("Urales")
's'
```

2. **Contar** (**recorrido**) Diseña una función **`contarLetra(str, letra)`** que, dado un texto representado en un string (**str**), cuente y devuelva el número de **letra** que contiene. Ejemplo_:

```
>>> contarLetra('patata caliente','a')
4
>>> contarLetra('Esplugues','a')
0
```

3. **Cambiar letra por otra**. Diseña una función **`cambia(s,c1,c2)`** que, dado un texto representado en un string (**s**), cambie el carácter **c1** por el **c2**. Ejemplo:

```
>>> cambia('patata','a','e')
'petete'
>>> cambia('camion','o','ó')
'camión'
>>> cambia('patata','e','i')
'patata'
```

4. Buscar posición. (**búsqueda**) Diseña una función **`pos_a(s, k)`** que reciba un string **s** y un entero no negativo **k** y devuelva la posición donde se encuentra la **k-ésima** letra **'a'** dentro de **s**, o -1 si el string **s** tiene menos de k letras **'a'**. Ejemplo:

```
>>> pos_a('', 1)
-1
>>> pos_a('hola', 2)
-1
>>> pos_a('ara', 3)
-1
>>> pos_a('lalaland', 3)
5
>>> pos_a('almendro', 1)
0
```

5. ***Cambiar_minMay***. Diseña una función **`cambia_minMay(str)`** que, dado un string **`str`** devuelva el string pero cambiando sus **vocales minúsculas** por **mayúsculas**.

```
>>> cambia_minMay('Casablanca 20 Madrid 30')
'CAsAblAncA 20 MAdrId 30'
>>> cambia_minMay('Valencia - Alicante en 2 horas')
'VAlEncIA - AlIcAntE En 2 hOrAs'
```

6. ***BorrarNúmerosString***. Diseña una función **`delNumbers(str)`** que entre un string **`str`** y devuelva el string pero sin sus caracteres numéricos o dígitos.

```
>>> delNumbers('Alicante a 20 kms')
'Alicante a  kms'
>>> delNumbers('Casa_1234, Bcn25')
'Casa_, Bcn'
```

7. **ContarMayusculas_minusculas**. Diseña una función **`contarMm(str)`** que cuente y devuelva el número de mayúsculas y minúsculas de un texto que entra como un string.

```
>>> contarMm('Hola')
(1, 3)
>>> contarMm('Ella dijo: HOLA')
(5, 7)
```

8. ***CuentaVocalesConsonantes***. Diseña una función que cuente y devuelva el número de vocales y consonantes de un texto que entra como un string.

```
>>> contar_vocal_cons('Aprendo Python')
(4, 9)
```

9. ***Encriptar***. Diseña una función **`encripta(s, clave)`** que reciba un string **`s`** con un mensaje y un string **`clave`** con una clave de codificación, y retorne el mensaje codificado según la clave leída. Los signos de puntuación y dígitos que aparezcan en el mensaje deben conservarse sin cambios. La clave consiste en una sucesión de las 26 letras minúsculas del alfabeto, las cuales se hacen corresponder con el alfabeto básico (a…z, sin la ñ o vocales con tilde) de 26 letras. La primera letra de la clave se relaciona con la letra **'a'**, la segunda con la letra **'b'**, etc. Por ejemplo, una entrada de la forma:
`Clave = 'ixmrklstnuzbowfaqejdcpvhyg'`
Texto para codificar: `'cafe'`
Con esta clave la letra **'i'** se corresponde con la letra **'a'**, la letra **'x'** se corresponde con la letra **'b'**, la letra **'m'** se corresponde con la letra **'c'**, y así sucesivamente. En el ejemplo anterior debería dar como salida: `'milk'`.
Nota: para simplificar consideraremos solo mensajes de entrada en minúsculas.

```
>>> encripta('cafe', 'ixmrklstnuzbowfaqejdcpvhyg')
'milk'
>>> encripta('dame 1 chocolate', 'ixmrklstnuzbowfaqejdcpvhyg')
'riok 1 mtfmfbidk'
```

10. ***Desencriptar***. Diseña una función **`desencripta(s, clave)`** que realice la función inversa a la función anterior, es decir, reciba un string **`s`** y una **clave** y realice la desencriptación del mensaje en el string devolviendo la cadena desencriptada.

```
>>> clave = 'ixmrklstnuzbowfaqejdcpvhyg'
>>> desencripta('milk',clave)
'cafe'
>>> desencripta('riok 1 mtfmfbidk',clave)
'dame 1 chocolate'
```

11. **Reagrupar**. Diseña una función `reagrupar(s)` que recibe como argumento una cadena (string) **`s`** que consta de letras minúsculas y dígitos, y devuelve una reagrupación de los caracteres de la cadena. La reagrupación es la siguiente: en la cadena devuelta, primero aparecen las letras, en el mismo orden en que vinieron en la cadena de argumentos; y luego vienen los dígitos, en el mismo orden en que vinieron en la cadena de argumentos. Ejemplo:

```
>>> reagrupar('r2b2')
'rb22'
>>> reagrupar('a45tr09pw')
'atrpw4509'
>>> reagrupar('nonumbers')
'nonumbers'
>>> reagrupar('543210')
'543210'
```

12. **Cola de personas**. Un string formado por **+** y **–** representa la evolución de una cola. Un signo **+** en el string significa que llega una persona nueva a la cola. Un **–** indica que se va una persona. Cada segundo una persona entra o sale de la cola, pero no hay entradas y salidas simultáneas. Un string como el descrito de tamaño **n** permite representar la evolución de la cola en los **n** primeros segundos. Por ejemplo, el string **`++-+++-`** representa la evolución de una cola durante siete segundos. En todos ellos se incorpora una persona salvo en los segundos tres y siete, en los que sale una. En su evolución el tamaño máximo de la cola ha sido cuatro.

    Está lloviendo y solo las tres primeras personas de la cola pueden ponerse a cubierto. Diseñad una función `t_descubierto(s)` que, dado un string `s` formado por los caracteres **+** y **–** que representa la evolución de la cola, nos diga durante cuántos segundos ha habido gente en la cola mojándose.

```
>>> t_descubierto('++++--+')
1
>>> t_descubierto('++++++++++----+')
12
>>> t_descubierto('+-++--++++--++--')
0
```

13. **Recorrido de robot**. Disponemos de un robot que se desplaza dando pasos de 60 centímetros siguiendo cuatro direcciones norte, sur, este y oeste. Las órdenes se transmiten al robot mediante un string formado por las letras **n**, **s**, **e** y **o** que responden a las cuatro direcciones en las que que el robot puede desplazarse. Diseña una función `vuelve(orden)` que dado un string `orden` formado con las letras **n**, **s**, **e** y **o** retorne **True** si y sólo sí el robot termina en la posición de salida. Si no, devuelve **False**. Ejemplo:

```
>>> vuelve('nseo')
True
>>> vuelve('nen')
False
```

14. **Contador de átomos**. Una fórmula química es una representación convencional de los elementos que forman un compuesto. Por ejemplo, el 1-2-butadiol sería C2H5O, que nosotros representaremos 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 `cuenta_atomos(s)` que, dado un string **s** con un compuesto como los descritos antes, devuelve el número de átomos que contiene. Para simplificar el problema, limitaremos el número que puede seguir el símbolo de un elemento a un valor entre **2** y **9**. Puede utilizar como métodos `str.isupper()`, etc. de la clase string.

```
>>> cuenta_atomos('HIO')
3
>>> cuenta_atomos('H2O')
3
>>> cuenta_atomos('C2H5O')
8
>>> cuenta_atomos('CaCO3')
5
>>> cuenta_atomos('Fe3O4')
7
```