UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ACG - Auditoría Calidad y Gestión de Sistemas

2024/2025

Francesco Lorenzoni
PCA25403GU

Practica 4 y 5

Pytest

Contents

Chapter 1

Práctica 4 y 5

1.1 Ejercicios 1/2/3 - `sin_vocales`

1.1.1 Ejercicio 1 - Función inicial

```
def sin_vocales (s):
    """
    devuelve el argumento s sin vocales
    """
    vocales = 'aeiou'
    s_sinVocales = ' '
    for ch in s:
        pos = vocales.find(ch)
        if pos == -1: #ch no es vocal
            s_sinVocales = s_sinVocales + ch
    return s_sinVocales
```

Listing 1.1: Función inicial

Esta es la función inicial que nos ha dado, y que inicialmente puede parecer funcionar, pero solo con pruebas demasiado sencillas.

1.1.2 Ejercicio 2 - Prueba de la función

Abajo en el código 1.3, después de la discusión de los dos errores encontrados, hay la función de prueba completa `test_sin_vocales()` que comprueba la función `sin_vocales(s)`.

El primero error en la función dada `sin_vocales(s)` es que no se tiene cuenta de las **mayúsculas** y **minúsculas**. Por lo tanto, la función no elimina las vocales mayúsculas. Para solucionarlo, es suficiente añadir a la lista de vocales la versión mayúscula de cada vocal.

```
def test_sin_vocales():
    s = "el agua esta mojada"
    exp = "l g st mjd"
    assert sin_vocales(s) == exp

    ...

    s = "El AgUe eStA mOjAdA"
    exp = "l g St mjd"
    assert sin_vocales(s) == exp
```

```
def sin_vocales (s:str):
    # vocales = 'aeiou'
    vocales = 'aeiouAEIOU'
    s_sinVocales = ''
    for ch in s:
        pos = vocales.find(ch)
        if pos == -1: #ch no es vocal
            s_sinVocales = s_sinVocales + ch
    return s_sinVocales
```

Listing 1.2: `sin_vocales()` con vocales mayúsculas

El segundo error es que la función no tiene en cuenta los **acentos**. Por lo tanto, la función no elimina las vocales acentuadas.

```
assert sin_vocales("áÉíÓú") == "" # falla
```

Para solucionarlo, es suficiente añadir a la lista de vocales la versión acentuada de cada vocal.

```
vocales = 'aeiouAEIOUáéíóúàèìòùäëïöüâê
îôûÁÉÍÓÚÀÈÌÒÙÄËÏÖÜÂÊÎÔÛãõÃÕ'
```

Esta solución no parece muy elegante, ya que la lista de vocales se vuelve muy larga. Buscando en internet he visto que una solución más elegante sería usar una expresión regular para eliminar todas las vocales, acentuadas o no. Para ello, se puede usar el módulo `re` de Python, junto con `unicodedata`, como se muestra en el código a la derecha.

```
import unicodedata
import re

def sin_vocales(s):
    # Normaliza el texto separando los
        caracteres básicos de sus diacríticos
    s_norm = unicodedata.normalize('NFD', s)
    # Elimina todas las vocales base y los
        diacríticos
    s_sin_vocales = re.sub(r'[aeiouAEIOU\u0300
        -\u036f]', '', s_norm)
    return s_sin_vocales
```

Listing 1.3: Código para eliminar vocales y diacríticos que utiliza `unicodedata`

```
def test_sin_vocales():
    s = "el agua esta mojada"
    exp = "l g st mjd"
    assert sin_vocales(s) == exp
    s = "aieou"
    exp = ""
    assert sin_vocales(s) == exp
    s = "123greee"
    exp = "123gr"
    assert sin_vocales(s) == exp
    s = "El AgUe eStA mOjAdA"
    exp = "l g St mjd"
    assert sin_vocales(s) == exp
    s = "àéíóú"
    exp = ""
    assert sin_vocales(s) == exp
    s = "are you ok?"
    exp = "r y k?"
    assert sin_vocales(s) == exp
    s = ""
    exp = ""
    assert sin_vocales(s) == exp
    s = "A"
    exp = ""
    assert sin_vocales(s) == exp
```

Listing 1.3: Pruebas que he escrito

1.1.3 Ejercicio 3/4 - Corregir la función y añadir pruebas

```
def sin_vocales (s:str):
    vocales = 'aeiouAEIOUáéíóúàèìòùäëïöüâêîôûÁÉÍÓÚÀÈÌÒÙÄËÏÖÜÂÊÎÔÛãõÃÕ'
    s_sinVocales = ''
    for ch in s:
        pos = vocales.find(ch)
        if pos == -1: #ch no es vocal
            s_sinVocales = s_sinVocales + ch
    return s_sinVocales
```

Listing 1.4: `sin_vocales()` corregida

Para evitar el uso del paquete adicional `unicodedata`, he preferido utilizar simplemente la lista de vocales que incluye también las vocales acentuadas.

He añadido al fichero `.py` la función que nos ha dado en el ejercicio, para comprobar que también esos tests pasaran.

1.2 Ejercicio 5 - cantidad_numeros

```python
import re
def cantidad_numeros_sencilla(s: str):
    """
    Esta función recibe una cadena de texto y devuelve la cantidad de números que contiene.
    N.B. números, no digitos!
    """
    return len(re.findall(r'\d+', s))
```

Listing 1.5: Solución sencilla

Esta primera solución funciona y es concisa, pero no tiene en cuenta los números decimales (como 12.34) y las notaciones científicas (como $12 \cdot 10^6$ o 10^{-2}). Para solucionarlo, es necesario modificar la expresión regular para que también considere estos casos. La manera elegida para hacerlo consiste en utilizar tres expresiones regulares diferentes, una para cada caso, eliminar las coincidencias de la cadena original y luego buscar los números restantes (enteros) en la cadena modificada, para finalmente contar el total de números encontrados.

```python
def cantidad_numeros(s: str):
    """
    Esta función recibe una cadena de texto y devuelve la cantidad de números que contiene.

    Números decimales (como 12.34) y notaciones científicas (como 12*10^6 o 10^(-2)) son
        considerados números.
    """
    # Identificar números decimales (como 12.34)
    decimal_pattern = r'-?\d+\.\d+'
    decimal_matches = re.findall(decimal_pattern, s)

    # Eliminar los números decimales ya encontrados para evitar contar dos veces
    for match in decimal_matches:
        s = s.replace(match, ' ', 1)

    # Identificar notaciones científicas (como 12*10^6 o 10^(-2))
    scientific_pattern = r'\d+\*10\^\(?\-?\d+\)?'
    scientific_matches = re.findall(scientific_pattern, s)

    # Eliminar las notaciones científicas de la cadena
    for match in scientific_matches:
        s = s.replace(match, ' ', 1)

    # Encontrar los números restantes (enteros)
    # Modificamos el patrón para capturar secuencias de dígitos en cualquier contexto
    remaining_pattern = r'\d+'
    remaining_matches = re.findall(remaining_pattern, s)

    # Contar el total de números encontrados
    return len(decimal_matches) + len(scientific_matches) + len(remaining_matches)
```

Listing 1.6: Solución que incluye números decimales y notaciones científicas

```
def test_cantidad_numeros():
    s = "12 356 53333"
    assert cantidad_numeros(s) == 3
    s = "asfa432asf23"
    assert cantidad_numeros(s) == 2
    s = ""
    assert cantidad_numeros(s) == 0
    s = "1"
    assert cantidad_numeros(s) == 1
    s = "fwgds"
    assert cantidad_numeros(s) == 0
    s = "1 fhsdSGG 4"
    assert cantidad_numeros(s) == 2
    s = "-12 + 34 -12-133 "
    assert cantidad_numeros(s) == 4
    s = "12.34 56.78"
    assert cantidad_numeros(s) == 2
    s = "12*10^6"
    assert cantidad_numeros(s) == 1
    s = "23*10^(-2)"
    assert cantidad_numeros(s) == 1
    s = "23*10^(64) + 12.34"
    assert cantidad_numeros(s) == 2
    s = "asgre23*10^(-2)asfg10^2"
    assert cantidad_numeros(s) == 2
    s = "asgre23*10^(-2)asfg10^2agr12.34"
    assert cantidad_numeros(s) == 3
```

Listing 1.7: Mis pruebas para `cantidad_numeros()`

```
def test_cantidad_numeros_doc():
    # Varios números en la cadena
    s = "un 1, un 201 y 2 unos"
    assert cantidad_numeros(s) == 3
    # Sin números en la cadena
    s = "sin numeros"
    assert cantidad_numeros(s) == 0
    # Un solo número en la cadena
    s = "2345543"
    assert cantidad_numeros(s) == 1
    # Diferentes longitudes de números
    s = "1 22 333 4444 55555"
    assert cantidad_numeros(s) == 5
    # Números separados por espacios
    s = "123 456 789"
    assert cantidad_numeros(s) == 3
    # Números separados por comas
    s = "12,34,56,78"
    assert cantidad_numeros(s) == 4
    # Números rodeados de texto
    s = "numero123numero456numero"
    assert cantidad_numeros(s) == 2
    # Una cadena continua de números
    s = "123456789"
    assert cantidad_numeros(s) == 1
    # Números al inicio y al final
    s = "1 starting and ending 2"
    assert cantidad_numeros(s) == 2
    # Número al final
    s = "ending with number 5"
    assert cantidad_numeros(s) == 1
    # Números en una cadena simple
    s = "3 6 9"
    assert cantidad_numeros(s) == 3
    # Sin números en la cadena
    s = "sinnumeros"
    assert cantidad_numeros(s) == 0
    # Números mezclados con texto
    s = "123unnumero456"
    assert cantidad_numeros(s) == 2
    # Números negativos
    s = "-5 10 -15"
    assert cantidad_numeros(s) == 3
```

Listing 1.7: Pruebas ordenadas como estan en el documiento

1.3 Ejercicio 6 - `sin_vocales` con prueba parametrizada

```
@pytest.mark.parametrize ("entrada , salida_esperada " ,[
   ("El agua esta mojada", "l g st mjd"),
   ("mojada bañando en el agua", "mjd bñnd n l g"),
   ("ahora termina bien", "hr trmn bn"),
   ("", ""),
   ("a", ""),
   ("m", "m"),
   ("unstringsinespacios", "nstrngsnspcs"),
   ("MAYUSculas FUNCIOnaN", "MYScls FNCnN"),
   ("krt yhgf dwpq", "krt yhgf dwpq"),
   ("aeoiuuuoiea", ""),
   ("disco de los 80", "dsc d ls 80"),
   ("signos como ? y ! y ¡", "sgns cm ? y ! y ¡"),
   ("ábc élla ó", "bc ll "),
   ("Óm tambien mayÚsculÁs", "m tmbn myscls")
   ])

def test_sin_vocales_parametrizado(entrada, salida_esperada):
   """
   testea la función sin_vocales
   """
   assert sin_vocales (entrada) == salida_esperada
```

Esta es una forma alternativa de escribir los tests, que permite parametrizar la función driver de test.

En comparación con el test set anterior del pdf, hemos añadido pruebas que incluyen vocales con acento. La función `sin_vocales` no va a cambiar porque ya había tenido en cuenta los acentos en el ejercicio anterior.