Práctica 2

March 29, 2022

1 Práctica 2 - Seminario de Lenguaje Python 2022

2 Ejercicios

2.1 Parte 1

1. Tomando el texto del `README.md` de numpy, imprima todas las líneas que contienen 'http' o 'https'.

Recordar: * split(): ¿qué caracter utilizarían para separar? * string **in** frase: para saber si un substring está incluido dentro de una frase * ¿podemos preguntar si un string se encuentra en una frase antes de recorrer la lista de palabras?

2. Indique la palabra que aparece mayor cantidad de veces en el texto del `README.md` de numpy. Recordemos algunas funciones de string:

[12]:
```
texto = """The constants defined in this module are:The constants defined in␣
 ↪this module are:

string.ascii_letters
The concatenation of the ascii_lowercase and ascii_uppercase constants␣
 ↪described below. This value is not locale-dependent.

string.ascii_lowercase
The lowercase letters 'abcdefghijklmnopqrstuvwxyz'. This value is not␣
 ↪locale-dependent and will not change.

string.ascii_uppercase
The uppercase letters 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'. This value is not␣
 ↪locale-dependent and will not change.
"""
# lista con todas las palabras
print(texto.lower().split())
# lista con palabras sin repetir
print(set(texto.lower().split()))
```

```
['the', 'constants', 'defined', 'in', 'this', 'module', 'are:the', 'constants',
'defined', 'in', 'this', 'module', 'are:', 'string.ascii_letters', 'the',
'concatenation', 'of', 'the', 'ascii_lowercase', 'and', 'ascii_uppercase',
```

```
'constants', 'described', 'below.', 'this', 'value', 'is', 'not', 'locale-
dependent.', 'string.ascii_lowercase', 'the', 'lowercase', 'letters',
"'abcdefghijklmnopqrstuvwxyz'.", 'this', 'value', 'is', 'not', 'locale-
dependent', 'and', 'will', 'not', 'change.', 'string.ascii_uppercase', 'the',
'uppercase', 'letters', "'abcdefghijklmnopqrstuvwxyz'.", 'this', 'value', 'is',
'not', 'locale-dependent', 'and', 'will', 'not', 'change.']
{'string.ascii_uppercase', 'value', 'concatenation', 'ascii_lowercase', 'is',
'are:', 'ascii_uppercase', 'constants', 'locale-dependent.', 'and',
'string.ascii_letters', 'uppercase', 'this', 'of', 'string.ascii_lowercase',
'below.', 'locale-dependent', 'change.', 'defined', 'lowercase', 'the',
'module', 'not', 'are:the', "'abcdefghijklmnopqrstuvwxyz'.", 'will', 'in',
'described', 'letters'}
```

Investigue el módulo Counter para simplificar la resolución.

2.1.1 Identificando mayúsculas, minúsculas y caracteres no letras

```
[ ]: caracter = "T"
print(texto.split()[0].startswith(caracter))
```

¿Pero qué pasa si queremos saber indistintamente si la palabra comienza con dicha letra en minúscula o mayúscula?

```
[ ]: caracter = "t"
print(texto.lower().split()[0].startswith(caracter))
```

¿Y si el caracter ingresado no es una letra?

```
[ ]: import string
caracter = "?"
print(f"El caracter es una letra {caracter in string.ascii_letters}")
```

3. Dado un texto solicite por teclado una letra e imprima las palabras que comienzan con dicha letra. En caso que no se haya ingresado una letra, indique el error. *Ver: módulo string*

4. Para la aceptación de un artículo en un congreso se definen las siguientes especificaciones que deben cumplir y recomendaciones de escritura:

- **título**: 10 palabras como máximo
- cada oración del **resumen**:
  - hasta 12 palabras: fácil de leer
  - entre 13-17 palabras: aceptable para leer
  - entre 18-25 palabras: difícil de leer
  - mas de 25 palabras: muy difícil

Dado un artículo en formato string, defina si cumple las especificaciones del título y cuántas oraciones tiene de cada categoría. El formato estándar en que recibe el string tiene la siguiente forma:

```
evaluar = """ título: Experiences in Developing a Distributed Agent-based Modeling Toolkit
with Python
resumen: Distributed agent-based modeling (ABM) on high-performance computing resources
provides the promise of capturing unprecedented details of large-scale complex systems.
However, the specialized knowledge required for developing such ABMs creates barriers to
wider adoption and utilization. Here we present our experiences in developing an initial
implementation of Repast4Py, a Python-based distributed ABM toolkit. We build on our
experiences in developing ABM toolkits, including Repast for High Performance Computing
(Repast HPC), to identify the key elements of a useful distributed ABM toolkit. We leverage
the Numba, NumPy, and PyTorch packages and the Python C-API to create a scalable modeling
system that can exploit the largest HPC resources and emerging computing architectures.
"""
```

En este ejemplo debería informar:

- titulo está ok
- Cantidad de oraciones fáciles de leer: 1, aceptables para leer: 2, dificil de leer: 1, muy dificil de leer: 2

Notas: * investigue Pattern Matching para una solución simplificada. * ¿cuántas variables utilizó para guardar la cantidad de cada categoría, se podría usar alguna estructura?

5. Dada una frase y un string ingresados por teclado (en ese orden), e informe la cantidad de veces que se encuentra el string en la frase. No distingir entre mayúsculas y minúsculas.

Ejemplo 1

- **Para la frase**: "Tres tristes tigres, tragaban trigo en un trigal, en tres tristes trastos, tragaban trigo tres tristes tigres."
- **Palabra**: "tres"
- **Resultado**: 3

Ejemplo 2

- **Para la frase**: "Tres tristes tigres, tragaban trigo en un trigal, en tres tristes trastos, tragaban trigo tres tristes tigres."
- **Palabra**: "tigres"
- **Resultado**: 2

Ejemplo 3

- **Para la frase**: "Tres tristes tigres, tragaban trigo en un trigal, en tres tristes trastos, tragaban trigo tres tristes tigres."
- **Palabra**: "TRISTES"
- **Resultado**: 3

2.2 Parte 2

5. Retomamos el código visto en la teoría, que informaba si los caracteres "@" o "!" formaban parte de una palabra ingresada

```
[ ]: cadena = input("Ingresa la clave (debe tener menos de 10 caracteres y no␣
      ↪contener los símbolos:@ y !):")
     if len(cadena) > 10:
         print("Ingresaste más de 10 carcateres")
     cant = 0
     for car in cadena:
         if car == "@" or car == "!":
             cant = cant + 1
     if cant >= 1:
         print("Ingresaste alguno de estos símbolos: @ o !" )
     else:
         print("Clave válida")
```

¿Cómo podemos simplificarlo?

6. Dada una frase donde las palabras pueden estar repetidas e indistintamente en mayúsculas y minúsculas, imprimir una lista con todas las palabras sin repetir y en letra minúscula.

```
frase = """
  Si trabajás mucho CON computadoras, eventualmente encontrarás que te gustaría
  automatizar alguna tarea. Por ejemplo, podrías desear realizar una búsqueda y
  reemplazo en un gran número DE archivos de texto, o renombrar y reorganizar
  un montón de archivos con fotos de una manera compleja. Tal vez quieras
  escribir alguna pequeña base de datos personalizada, o una aplicación
  especializada con interfaz gráfica, o UN juego simple.
  """
```

7. Escriba un programa que solicite que se ingrese una palabra o frase y permita identificar si la misma es un Heterograma (tenga en cuenta que el contenido del enlace es una traducción del inglés por lo cual las palabras que nombra no son heterogramas en español). Un Heterograma es una palabra o frase que no tiene ninguna letra repetida entre sus caracteres.

Tener en cuenta - Lo que no se puede repetir en la frase son sólo aquellos caracteres que sean letras. - No se distingue entre mayúsculas y minúsculas, es decir si en la frase o palabra tenemos la letra “T” y la letra “t” la misma NO será un Hererograma. - Para simplificar el ejercicio vamos a tomar como que las letras con tilde y sin tilde son distintas. Ya que Python las diferencia:

```
>>> 'u' == 'ú'
False
```

Ejemplos

Entrada	¿Heterograma?
cruzamiento	Sí
centrifugados	Sí
portón	Sí
casa	No
día de sol	No
con diez uñas	Sí
no-se-duplica	Sí

8. Escriba un programa que solicite por teclado una palabra y calcule el valor de la misma dada la siguiente tabla de valores del juego Scrabble:

Letra	valor
A, E, I, O, U, L, N, R, S, T	1
D, G	2
B, C, M, P	3
F, H, V, W, Y	4
K	5
J, X	8
Q, Z	10

*Tenga en cuenta qué estructura elige para guardar estos valores en Python

Ejemplo 1

- **Palabra**: "solo"
- **valor**: 4

Ejemplo 2

- **Palabra**: "tomate"
- **valor**: 8

9. La idea es tratar de programar una de las partes principales del juego "Buscaminas". La idea es que dado una estructura que dice que celdas tienen minas y que celdas no las tienen, como la siguiente:

```
[
  '-*-*-',
  '--*--',
  '----*',
  '*----',
]
```

Generar otra que indique en las celdas vacías la cantidad de bombas que la rodean, para el ejemplo anterior, sería:

```
[
  '1*3*1',
  '12*32',
  '1212*',
  '*1011',
]
```

Nota: Defina al menos una función en el código (si hay mas mejor) y documente las mismas con **docstring** que es lo que hacen.

10. Trabajando con los contenidos de los archivos que pueden acceder en el curso:

- nombres
- eval1

- eval2

Manipule estos archivos para realizar lo siguiente:

- Generar una estructura con los nombres de los estudiantes y la suma de ambas notas.
- Calcular el promedio de las notas totales e informar que alumnos obtuvieron menos que el promedio.

11. Con la información de los archivos de texto que se encuentran disponibles en el curso:

- nombres_1
- nombres_2
- Indique los nombres que se encuentran en ambos. **Nota**: pruebe utilizando list comprehension.
- Genere tres variables con la lista de notas y nombres que se incluyen en los archivos: *nombres_1*, *eval1.txt* y *eval2.txt* e imprima con formato los nombres de los estudiantes con las correspondientes nota y la suma de ambas como se ve en la imagen

3 Segunda entrega

- **Puntos:** 20.
- **Fecha límite de entrega:** Miércoles 13 de Abril 23:59hs.
- **Modalidad de entrega:**
  1. Elige uno entre los ejercicios 10 y 11 que resolviste en esta práctica.
  2. Publica el código de la resolución en un cuaderno de Jupyter Notebook en tu repositorio creado en Github Github. Explicar en celdas con formato Markdown el significado de los passos principales realizados para la resolución.
  3. Agregar el link del repositorio a la tarea en el curso.

[ ]: