

UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA



PROYECTO FINAL SEMINARIO OPCIÓN PYTHON

**Desarrollo con Raspberry Pi y sensores.
Desarrollo educativo destinado a escuelas primarias:
Sopa de letras**

Autores

Juan Pablo Sánchez Magariños

Bruno Sbrancia

Matías Agustín Cabral

7 de octubre de 2019

Índice general

1. Introduction	1
2. Temas investigados	2
2.1. Sobre PySimpleGUI	2
2.2. Sobre Pattern	2
2.2.1. <code>pattern.web</code> :	2
2.2.2. <code>pattern.es</code>	3
2.3. Sobre el trabajo con sensores	3
3. Implementación	4
3.1. Grilla	4
3.2. Configuración	4
3.3. Sopa	5
3.4. Problemas encontrados	5
3.4.1. Wiktionary	5
3.4.2. Pattern	6
3.4.3. Deshabilitar ventana	7
3.5. Problema de combos duplicados	7
4. Conclusión	9
4.1. Trabajo a futuro	9
Bibliografía	11
A. Etiquetado en Pattern	12
B. Raspberry Py	23
B.1. Raspeberry Pi	23
B.1.1. Primer aproximación	23
B.1.2. Probando el sistema Raspbian	23
B.1.3. Instalación de requisitos	23
B.1.4. Usando los sensores y dispositivos	24
B.1.5. Por ejemplo, para simular dos modulos de 8x8 en matriz de led en tiempo real:	27

1. Introduction

Este trabajo se lleva a cabo en el marco de la asignatura Seminario de Lenguajes Como trabajo final para aplicar los conocimientos adquiridos durante la cursada, se implementa una aplicación didáctica orientada al desarrollo de capacidades de lecto-comprensión al mismo tiempo que permite un esparcimiento lúdico

Lo que a continuación se presenta es la recopilación de las diferentes fases del proceso de desarrollo, indicando y explicando el lenguaje de programación y los módulos utilizados para tal fin.

Se incluye también un detalle de las problemáticas y obstáculos que se debieron superar,

2. Temas investigados

2.1. Sobre PySimpleGUI

Es un paquete de GUI, (Interfaz Gráfica de Usuario) pensada para principiantes. Los paquetes GUI con más funcionalidad, como QT y WxPython, requieren configuración y tiempo para familiarizarse con las interfaces. Con PySimpleGUI se pueden crear GUI personalizadas de manera que se acopla fácilmente a lo que se aprende en las primeras aproximaciones a Python. La disposición (*layout*) de la GUI es una lista de filas. Cada fila es una lista de elementos. Cuando vuelve la llamada para mostrar el formulario, todos los campos de entrada en el formulario se devuelven como una tupla que se puede evaluar para llevar a cabo las distintas respuestas del sistema.

Se basa en tkinter, por lo que no hay otras dependencias de paquetes.

2.2. Sobre Pattern

Pattern es un paquete para Python 2.4+ con funcionalidades para minería web (Google, Twitter, Wikipedia), procesamiento de lenguaje natural, aprendizaje automático y análisis de red. El código fuente [1] tiene licencia bajo BSD.

Está organizado en módulos separados que se pueden encadenar juntos. Por ejemplo, el texto de Wikipedia (**pattern.web**) se puede analizar para etiquetado gramatical (**pattern.es**), consultado por sintaxis y semántica (**pattern.search**), y utilizado para entrenar a un clasificador (**pattern.vector**). En el trabajo utilizamos principalmente **pattern.web** y **pattern.es**.

2.2.1. pattern.web:

Herramientas para la minería de datos web, utilizando un mecanismo de descarga que admite almacenamiento en caché, servidores proxy, solicitudes asíncronas y redirección. Una clase **SearchEngine** proporciona una API uniforme para múltiples servicios web: Google, Bing, Yahoo!, Twitter, Wikipedia, Flickr y fuentes de noticias utilizando **feedparser** [2]. El módulo incluye un analizador HTML basado en **BEAUTIFUL SOUP** [3], un analizador PDF basado en **PDFMINER** [4], un rastreador web y una interfaz de correo web.

2.2.2. **pattern.es**

Contiene un etiquetador gramatical (identifica sustantivos, adjetivos, verbos, etc. en una oración), herramientas para la conjugación de verbos, la singularización y pluralización de sustantivos en Español.

2.3. **Sobre el trabajo con sensores**

Se usa la raspberry para medir temperatura y bla bla para detalles de como implementarlo detalladamente ver el [anexoraspberry]

3. Implementación

Como primer paso pensamos que para poder implementar la aplicación de la sopa de letras sería conveniente tener, en principio un módulo que genere y maneje cierta *grilla* o *matriz* con las letras correspondientes, otro módulo que muestre en pantalla dichos datos y otro que le sirva al usuario para configurar los primeros.

3.1. Grilla

Para el primero que decidimos llamar `grilla.py` empezamos por leer distintas implementaciones de juegos similares. En Rosetta Code [[rosetta](#)], encontramos una solución para la *busqueda de palabras*, que es parecido salvo porque en este caso las letras que sobran, es decir las que no pertenecen a ninguna palabra de las buscadas, deben formar un *mensaje oculto*.

En resumen como se puede interpretar en el código [[anexogrilla](#)] se recibe una lista de palabras, para cada una elige aleatoriamente una posición en la grilla, y una dirección (de un conjunto de *versores* posibles) y comprueba si cabe de esa manera. Para ello, para cada letra prueba que no se salga de los bordes de la grilla, y que la celda este: libre u ocupada con la misma letra que se esta probando. De no ser así ira “avanzando” en la grilla hasta haber probado todos los casilleros de la misma como celda de comienzo para cada palabra. Este proceso lo limitamos con un número de iteraciones máximo para que no entre en un bucle infinito en el caso que se de una situación imposible de resolver. En cada iteración se mezcla la lista de palabras para comenzar con una distinta. Finalizado el proceso obtendremos una matriz cuyos elementos son diccionarios con campos para indicar distintas propiedades como la letra, si es parte de la solución, de qué tipo es la palabra a la que pertenece, si pertenece a más de una palabra, etc.

3.2. Configuración

En el modulo `config.py` se recopilan todos los datos necesarios para la confección del juego, y se guarda todo en un archivo *json*. También en el se analizan las palabras propuestas por el usuario para ser parte del conjunto solución. Se buscan las palabras en Wiktionary en Pattern para obtener información sobre su definición y categoría, y se evalúan estos resultados, generando un reporte de error en caso de discrepancia. Los detalles de esta implementación se mencionan en las secciones [3.4.1](#) y [3.4.2](#).

También se establecen los colores de la interfaz en general a partir de información de temperatura y humedad obtenidos con la RaspberryPy (ver apéndice B).

Para asegurarse que se distingan los colores elegidos para resaltar las leras de cada tipo de palabras, usamos un modulo llamado `colormath` que asigna un valor numérico a los colores y calcula su cercanía.

3.3. Sopa

Este módulo es el encargado de dibujar en pantalla la interfaz de usuario. Para ello recibe una configuración de un archivo (cuya generación se explica en la sección 3.2) dónde se le indicaran sus propiedades, como los colores que se deben usar, la fuente, las palabras de la solución, etc.

A partir de la lista de palabras define el tamaño de la grilla, calculando el máximo entre el tamaño de la palabra más larga y la cantidad de palabras.

Luego utilizando el módulo `PySimpleGui` representamos la grilla con una matriz de elementos `sg.Button`, que el usuario luego podrá presionar para marcar una letra. Para ello antes seleccionará un color dependiendo del tipo de palabra que busque.

En cada iteración del bucle de eventos, se comprueba si se ha llegado a la solución. El módulo `Win.Condition` comprueba celda por celda que estén marcadas correctamente, es decir si una celda no pertenece a una palabra, no debe estar marcada, si pertenece a una, debe estar marcada con el color correspondiente, y si pertenece a más de una puede estar marcada con cualquiera de los mismos. Además el usuario dispone de un botón para llamar a dicho módulo en cualquier momento, y se le indicará resaltando los errores cometidos.

3.4. Problemas encontrados

3.4.1. Wiktionary

Para extraer la definición de Wiktionary nos topamos con el problema de que al ser una página comunitaria no siempre obedece el mismo patrón la confección el artículo de una palabra.

Se nos ocurrió entonces ver el código fuente de las paginas donde notamos que para enumerar las definiciones de una palabra se usan siempre *HTML Description Lists*, aunque sea una sola. Por lo tanto decidimos que la mejor forma era buscar el código fuente este ítem:

3. Implementación

```
from pattern.web import Wiktionary, plaintext
engine = Wiktionary(language="es")
sch=engine.search(palabra)
if sch != None:
    pos_1 = sch.source.find('<dt>1</dt>')
    if pos_1 == -1:
        pos_1 = sch.source.find('<dt>')
        pos_cierre_1 = sch.source.find('</dt>',pos_1+1) #busca
        ↪ a partir de pos 1
    else:
        pos_cierre_1 = pos_1
```

Como a veces el ítem solo contiene el número 1 y a veces lo acompaña una palabra, se salva ese caso buscando la apertura `<dt>` y el cierre `</dt>` por separado. Así se obtiene una posición, que se utilizara luego como comienzo para extraer el texto.

De manera similar se obtiene la posición final, se puede buscar la apertura del siguiente ítem, y así nos quedaríamos siempre con la primer definición. Pero en caso que tenga una sola definición no se encontrará otra apertura, por lo tanto en el caso que devuelva negativa la búsqueda se puede buscar el siguiente punto. Luego con el módulo `plaintext` se traduce el código html

```
definicion = plaintext(sch.source[pos_cierre_1: pos_2])[:pos_punto+1]
if definicion[:1] == '1':
    definicion = definicion[1:pos_punto+1]
```

3.4.2. Pattern

Con pattern el primer problema que encontramos fue que es incompatible con Python3.7, que era el que teníamos instalado, y por lo tanto decidimos trabajar con Python 3.6.

A la hora de usar la función `tag` que sirve para etiquetar gramaticalmente las palabras de un *string* notamos que los casos que sea una cadena sin significado se le asigna la etiqueta `'NN'`, que debería corresponder a un sustantivo.

Se solucionó al darnos cuenta que el paquete contiene extensas listas de palabras de donde obtiene la información, por lo tanto importándolas y buscando en ellas antes de aplicar la función de etiquetado obtuvimos buenos resultados. Se presenta esta investigación en exhaustivo detalle en el apéndice [A](#)

3.4.3. Deshabilitar ventana

Nos sucedió en diferentes ocasiones en las que surgía un *pop up*, si se clickeaba otra ventana, se perdía el foco del primero y esto generaba errores, ya que el sistema podía quedar esperando cierto dato por ejemplo de un input. Para eso decidimos buscar un método para bloquear la interacción con las ventanas cuando fuera conveniente, pero lamentablemente el método proporcionado por PySimplegui para dicho fin, no nos funcionó. Por lo tanto decidimos redefinirlo de la siguiente manera:

```
def disable(window):
    window.TKroot.attributes('-disabled', 1)
    window.SetAlpha(0.75)

def enable(window):
    window.Reappear() ##igual que poner el alpha en 1
    window.BringToFront()
    window.TKroot.attributes('-disabled', 0)
```

Lo que hace es acceder directamente a los atributos del elemento heredado de `tkinter` y modificarlo para que se deshabilite toda la ventana. Además hacemos que la ventana bloqueada se aclare para que llame la atención la que está habilitada.

3.5. Problema de combos duplicados

Ya habiendo finalizado el desarrollo mas grueso de la aplicación, pasamos a una etapa de pruebas en donde nos percatamos de un error que se generaba en la ocasión especial que desde el menú se abriera la *configuración* por segunda vez consecutiva, sin cerrar el programa principal.

```
Traceback (most recent call last):
  File "run.py", line 56, in <module>
    config.main()
    .
    .
    .

  File "C:\~\Python36\lib\site-packages\PPySimpleGUI\PySimpleGUI.py",
line 5327, in PackFormIntoFrame
    combostyle.element_create(unique_field, "from", "alt")
  File "C:\~\Python36\lib\tkinter\ttk.py", line 468, in element_create
    spec, *opts)
_tkinter.TclError: Duplicate element _CANT_S_.TCombobox.field
```

3. Implementación

Luego de una extensa investigación, resolvimos que el error se generaba por el modo que tiene PySimpleGUI de manejar los elementos `combobox`. En el código donde define dichos elementos los llama

```
# Creates a unique name for each field element(Sure there is a better  
↪ way to do this)  
unique_field = str(element.Key) + '.TCombobox.field'
```

El problema es que este identificador no será único en el caso que se corra el código por segunda vez, sin antes *destruirlos* para que al correr otra vez el código, se creen nuevamente sin haber problema de nombres duplicados. La primer idea fue buscar alguna manera de hacer esto último desde los métodos de ventana del mismo PySimpleGUI pero no se logró encontrar dicha funcionalidad. Por cuestión de tiempo se decidió modificar el código fuente de PySimpleGUI donde define este campo unico anexandole el tiempo actual usando el paquete `time`:

```
unique_field = str(time.time()).replace('.', '') + str(element.Key) +  
↪ '.TCombobox.field'
```

Es una solución poco elegante, ya que se debe tener todo el código de PySimpleGUI en el directorio de ejecución, para que a la hora del `import`, se “pise” el paquete por la version modificada, pero fue la más solida que pudimos encontrar.

4. Conclusión

Como conclusión, queremos decir que a lo largo del trabajo hemos podido incorporar conocimientos, recursos y herramientas que consideramos de gran valor para nuestra formación y para el desarrollo de futuros proyectos. En primer lugar, por su misma extensión, el trabajo hizo necesaria una división de tareas en la que cada miembro del grupo tuvo que asumir un rol y trabajar para el conjunto. En este sentido, fue de gran utilidad el uso de la plataforma GitHub como herramienta de versionado de código.

Por otra parte, el uso de un lenguaje *open source* nos dio la posibilidad de capitalizar el trabajo realizado por otros programadores, cuyos códigos se encuentran disponibles en la web, adaptándolo a nuestros requerimientos, y asimismo poner nuestro propio trabajo a disposición de cualquiera que lo desee, ya sea para usar la aplicación, o para servirse del código.

Otro beneficio que obtuvimos del desarrollo de esta tarea, es haber aprendido a documentar debidamente un código, lo cual favorece y facilita su reutilización .

4.1. Trabajo a futuro

Por último, hacemos mención de una serie de aspectos que se podrían mejorar, o bien implementar para una versión futura de la aplicación:

- Embellecimiento de la interfaz:
 - Gráficas más amigables ("splash art")
 - Tachado de las palabras encontradas durante el juego
 - Mejora de los botones de selección de color para cada tipo de palabra durante el juego
 - Incorporación de efectos sonoros.
- Implementar una opción que permita salvar y cargar una partida.
- Incremento de la cantidad máxima de palabras permitidas para buscar en la sopa.
- Que el tamaño de los botones en que aparecen los botones de la sopa varíen su tamaño en función de la cantidad de palabras a encontrar (a mayor cantidad, menor tamaño).

4. Conclusión

- Solucionar la portabilidad de la aplicación empaquetando los recursos necesarios en un ejecutable.

Bibliografía

- [1] CLiPS. (). Sitio oficial de pattern, dirección: <http://www.clips.ua.ac.be/pages/pattern>.
- [2] (2004-2008). Feedparser 5.2.0 documentation, dirección: <https://packages.python.org/feedparser>.
- [3] (2004-2015). BeautifulSoup documentation, dirección: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.
- [4] (2004-2014). Pdfminer documentation, dirección: <https://euske.github.io/pdfminer/index.html>.
- [5] W. De Smedt T. Daelemans, «Pattern for python», *Journal of Machine Learning Research* 13 (2012) 2063-2067,
- [6] Clips. (2017). Pattern-es github repository, dirección: <https://github.com/clips/pattern/tree/master/pattern/text/es>.
- [7] B. K. Bul, *Theory principles and design of magnetic circuits*. Energia Press, 1964, pág. 464, (in Russian).
- [8] J. Breckling, ed., *The analysis of directional time series: Applications to wind speed and direction*, ép. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.
- [9] B. D. Cullity, *Introduction to magnetic materials*. Reading, MA: Addison-Wesley, 1972.
- [10] R. M. A. Dawson, Z. Shen, D. A. Furst, S. Connor, J. Hsu, M. G. Kane, R. G. Stewart, A. Ipri, C. N. King, P. J. Green, R. T. Flegal, S. Pearson, W. A. Barrow, E. Dickey, K. Ping, C. W. Tang, S. V. Slyke, F. Chen, J. Shi, J. C. Sturm y M. H. Lu, «Design of an improved pixel for a polysilicon active-matrix organic LED display», en *SID Tech. Dig.* 1998, vol. 29, págs. 11-14.
- [11] W. Dai, H. V. Pham y O. Milenkovic, «Distortion-rate functions for quantized compressive sensing», en *IEEE Information Theory Workshop on Networking and Information Theory*. 2009.
- [12] ———, «Comparative study of quantized compressive sensing schemes», en *IEEE Information Theory Workshop on Networking and Information Theory*. 2009.
- [13] S. G. Finn, M. Médard y R. A. Barry, «A novel approach to automatic protection switching using trees», **presented at** IEEE International Conference on Communications, Montreal, Que., Canada, 1997.

A. Etiquetado en Pattern

A la hora de usar la función `tag` que sirve para etiquetar gramaticalmente las palabras de un *string* notamos que los casos que sea una cadena sin significado se le asigna la etiqueta `'NN'`, que debería corresponder a un sustantivo. Veamos un ejemplo:

```
import pattern.es
import random
import string

x= "danzar amigable barrilete asdfg"
print(pattern.es.tag(x, tokenize=True, encoding='utf-8'))

x = ''.join([random.choice(string.ascii_letters + string.digits) for n
↪ in range(32)])
print(pattern.es.tag(x, tokenize=True, encoding='utf-8'))
```

Esto nos imprime:

```
[('danzar', 'VB'), ('amigable', 'JJ'), ('barrilete', 'NN'), ('asdfg', 'NN')]
[('KD1mrZi6SwpZN25arEGv4TugHLAy2BYC', 'NN')]
```

Las primeras tres palabras las etiqueta correctamente como verbo, adjetivo y sustantivo, pero ya en la siguiente vemos algo raro. Sin encontrar documentación al respecto procedemos a revisar el repositorio [pattern.es](#) donde notamos el archivo `es-spelling.txt`, el cual tiene un diccionario gigantesco con palabras como clave y un valor numérico por ejemplo `'ultravioleta': 5`. No se indica que significa el valor, entonces buscamos donde lo usa. En el archivo `_init_.py` define la variable `spelling`:

```
spelling = Spelling(
    path = os.path.join(MODULE, "es-spelling.txt")
)
```

que luego se usa para definir una [función](#) que sugiere correcciones. La clase `Spelling` la importa de `pattern.text`:

```
# Import spelling base class.
from pattern.text import (
    Spelling
)
```

En el `__init__.py` de `pattern.text` podemos investigar la [clase](#), dónde notamos el comentario:

```
# Based on: Peter Norvig, "How to Write a Spelling Corrector",
↳ http://norvig.com/spell-correct.html
```

Siguiendo el [link](#):

```
WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N
```

Podemos deducir que el *diccionario* de `es-spelling.txt` es en realidad un [Counter](#). Por lo tanto el numerito de los `values()` es un contador de la cantidad de veces que apareció la palabra en el entrenamiento que tuvo el paquete, como decía al principio:

```
;;; Based on several public domain books from Project Gutenberg
;;; and Wikipedia articles and online Spanish newspaper articles.
```

Otro archivo importante que se encuentra en *pattern.es*, es el [lexicon](#). Es tan grande que el visor de github no lo carga y hay que verlo en [raw](#). En él vemos que tiene palabras junto con lo que parece ser el *part_of_speech*, por ej `ballet NCS`. Si buscamos la referencia a NCS. en la [lista de abreviaciones](#) notamos que no aparece allí. Tampoco en ninguno de estos [tag sets](#). El único resultado alegórico lo encontramos en éste [paper](#) donde la sigla hace referencia al termino en inglés *Noun Compounds*, o sea

sustantivos compuestos, lo cual en mi opinión `ballet` no es pero bueno sigamos mirando que más tiene **lexicon**

En el `__init__.py` se usa:

```
lexicon = parser.lexicon # Expose lexicon.
```

A. Etiquetado en Pattern

Por lo tanto es otro atributo del paquete. Probamos en la línea de comandos que imprime lo siguiente:

```
>>> import pattern.es
>>> print(type(pattern.es.spelling))
<class 'pattern.text.Spelling'>
>>> print(dir(pattern.es.spelling))
['CYRILLIC', 'LATIN', '__class__', '__contains__',
↳ '__delattr__', '__delitem__', '__dict__', '__dir__', '__doc__',
↳ '__eq__', '__format__', '__ge__', '__getattribute__',
↳ '__getitem__', '__gt__', '__hash__', '__init__',
↳ '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
↳ '__module__', '__ne__', '__new__', '__reduce__',
↳ '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
↳ '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
↳ '_edit1', '_edit2', '_known', '_lazy', '_path', 'alphabet',
↳ 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys',
↳ 'language', 'load', 'path', 'pop', 'popitem', 'setdefault',
↳ 'suggest', 'train', 'update', 'values']
>>>
```

Notamos que tiene el método `keys()`, lo mismo sucede con `pattern.es.lexicon`, entonces probamos este pequeño programa: El resultado de LISTING se puede ver en la FIGURA

```
import pattern.es
c = 0
for x in pattern.es.lexicon.keys():
    if not( x in pattern.es.spelling.keys() ):
        print(x, end=', ')
        c += 1
print("\n")
print('Cantidad de palabras en lexicon que NO estan en spelling: ',c)
```



```

C:\Windows\system32\cmd.exe
imo, insula, integra, integramente, integro, integros, inter, inter-territorial,
intima, intimamente, intimas, intimo, intimos, istmica, italo-gótico, item, ite
ms, ile, ð, ðagastir, ñ, ñame, ñandúes, ñoquis, ó, óbitos, óblast, óculo, óculos
, ódenes, óleo, óleos, omnibus, ónice, ópera, óperas, óptica, ópticas, óptico, ó
ptico-isoméricas, ópticos, óptima, óptimas, óptimo, óptimos, órbita, órbitas, ór
denes, órgano, órganos, organum, origen, ósea, óseas, óseo, óseos, ósmos
is, óxido, óxido-reducción, óxidos, ò, úlceras, última, últimamente, últimas, úl
timo, últimos, única, únicamente, únicas, único, únicos, úrico, útero, útil, úti
les, últimos, úber, úlcha-domo, yr, ;), :-(-, :-(-, ;-), <3, <333, =), ♥, e, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00

Cantidad de palabras de lexicon que NO están en spelling: 69648

-----
(program exited with code: 0)
Presione una tecla para continuar . . .

```

Siguiendo con esa idea probamos con spelling:

```

c = 0
for x in pattern.es.spelling.keys():
    if not( x in pattern.es.lexicon.keys() ):
        print(x, end=', ')
        c += 1
print("\n")
print('Cantidad de palabras de spelling que no estan en lexicon: ',c)

```

```

C:\Windows\system32\cmd.exe
, zapatear, zapateos, zapatetas, zapaticos, zapatilla, zapecada, zaporozhye, zap
ota, zaque, zaques, zaquizam, zaragoza, zarandajas, zarandas, zarandeo, zarato,
zaraza, zarif, zaro, zaros, zarp, zarpasen, zarra, zarzas, zarzo, zas, zatrix, z
avhan, zcate, zcoos, zdal, zedong, zeffiretto, zegarra, zelador, zelanda, zelaya
, zembla, zemski, zenha, zero, zerrizuela, zeta, zeus, zeuxis, zgase, zguenlo, z
ha, zhambyl, zheng, zhnev, zhou, zhu, zhukov, ziganda, zigiotto, zimbabue, zimba
list, zimmerit, zinnemann, ziu, zoca, zocodover, zod, zoelas, zoetemelk, zole, z
ollinger, zolo, zoltan, zome, zone, zonzorino, zool, zopilote, zopiro, zoraida,
zordon, zoro, zoroastes, zoroastrismo, zorras, zorrilla, zorruna, zose, zoster,
zov, zquez, zuata, zubin, zubizarreta, zuccarini, zuecos, zug, zugarramurdi, zuh
ari, zuikaku, zulema, zulfikar, zulia, zum, zumbido, zumpango, zurda, zurich, zu
rr, zuzaban, zver,

Cantidad de palabras de spelling que NO están en lexicon: 24348

-----
(program exited with code: 0)
Presione una tecla para continuar . . .

```

Y por último la intersección:

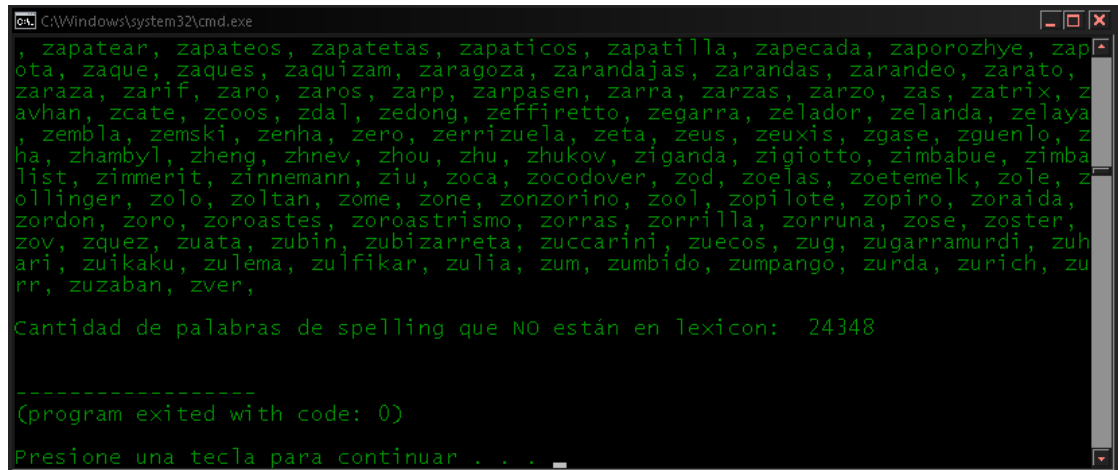
```

c = 0
for x in pattern.es.spelling.keys():
    if x in pattern.es.lexicon.keys():
        print(x, end=', ')

```

A. Etiquetado en Pattern

```
c += 1
print("\n")
print('Cantidad de palabras de spelling que TAMBIÉN están en lexicon:
↪ ', c)
```



```
C:\Windows\system32\cmd.exe
, zapatear, zapateos, zapatetas, zapaticos, zapatilla, zapecada, zaporozhye, zap
ota, zaque, zaques, zaquizam, zaragoza, zarandajas, zarandas, zarandeo, zarato,
zaraza, zarif, zaro, zaros, zarp, zarpasen, zarra, zarzas, zarzo, zas, zatrix, z
avhan, zcate, zcoos, zdal, zedong, zeffiretto, zegarra, zelador, zelanda, zelaya
, zembla, zemski, zenha, zero, zerrizuela, zeta, zeus, zeuxis, zgase, zguenlo, z
ha, zhambyl, zheng, zhnev, zhou, zhu, zhukov, ziganda, zigiotto, zimbabue, zimba
list, zimmerit, zinnemann, ziu, zoca, zocodover, zod, zoelas, zoetemelk, zole, z
ollinger, zolo, zoltan, zome, zone, zonzorino, zool, zopilote, zopiro, zoraida,
zordon, zoro, zoroastes, zoroastrismo, zorras, zorrilla, zorruna, zose, zoster,
zov, zquez, zuata, zubin, zubizarreta, zuccarini, zuecos, zug, zugarramurdi, zuh
ari, zuikaku, zulema, zulfikar, zulia, zum, zumbido, zumpango, zurda, zurich, zu
rr, zuzaban, zver,

Cantidad de palabras de spelling que NO están en lexicon: 24348

-----
(program exited with code: 0)
Presione una tecla para continuar . . .
```

Como vemos son archivos disjuntos, lexicon tiene más símbolos, pero spelling también tiene palabras en otros idiomas y casos extraños, por ejemplo, no está ‘zanahoria’ pero sí ‘zanahorias’. Busquemos las entradas más largas de cada archivo:

```
lista = list(pattern.es.spelling.keys())
mas_largo = max(lista, key=len)
print(mas_largo, ' - ', len(mas_largo))
```

Resulta:

```
bienintencionadamente - 21
```

Con lexicon obtenemos:

```
..... - 71
```

Si contamos los distintos valores del diccionario:

```
from collections import Counter
c = Counter(pattern.es.lexicon.values())

print(c)
```

Esto devuelve:

```
Counter({'NP': 23462, 'NCS': 13780, 'AQ': 11017, 'VMI': 9016,
        'NCP': 7568, 'Z': 5969, 'VMP': 5259, 'VMN': 2206, 'RG': 1731,
        'VMS': 1485, 'VMG': 1258, 'NC': 1082, 'Zu': 441, 'W': 295,
        'VMM': 249, 'I': 161, 'Zp': 130, 'SP': 111, 'DI': 74, 'SYM': 63,
        'Zm': 59, 'AO': 56, 'VAI': 54, 'Fz': 48, 'PP': 42, 'DD': 29,
        'VSI': 27, 'Zd': 25, 'DP': 25, 'PI': 21, 'CC': 21, 'PD': 20,
        'PR': 16, 'PT': 16, 'CS': 15, 'VAS': 15, 'DA': 13, 'VSS': 9,
        'Fs': 6, 'PX': 6, 'Fe': 4, 'Fc': 3, 'VAG': 3, 'VAN': 3, 'Fa': 2,
        '":': 2, 'Fg': 2, 'Fh': 2, 'Fr': 2, 'Fi': 2, 'DT': 2, 'RN': 2,
        'PO': 2, 'VSN': 2, 'VSG': 2, 'Fpa': 1, 'Fpt': 1, 'Fp': 1, 'Fd': 1,
        'Fx': 1, 'VSP': 1, 'Fl': 1})
```

Por lo que vuelve a surgir la duda de qué significaran esos *tags* ya que no se encuentran en la [tabla](#) proporcionada por pattern. Viendo la definición de [parse](#):

```
parse(string,
      tokenize = True,           # Split punctuation marks from words?
      tags = True,              # Parse part-of-speech tags? (NN, JJ, ...)
      chunks = True,            # Parse chunks? (NP, VP, PNP, ...)
      relations = False,        # Parse chunk relations? (-SBJ, -OBJ, ...)
      lemmata = False,          # Parse lemmata? (ate => eat)
      encoding = 'utf-8'        # Input string encoding.
      tagset = None)            # Penn Treebank II (default) or UNIVERSAL.
```

UNIVERSAL se refiere a ese otro tagset que vemos en **lexicon**. Veamos qué palabras de ese conjunto no se corresponden con el etiquetado que proporciona la función `tag`. La misma recibe un *string* como parámetro, lo separa en palabras y devuelve una lista con las tuplas (palabra, etiqueta), (por eso accedemos con `[0][1]`):

```
for x in lexicon:
    if lexicon[x] != tag(x, tokenize=True, encoding='utf-8', tagset
        ↪ = 'UNIVERSAL')[0][1]:
        print(x, end=', ')
```

A. Etiquetado en Pattern

[illegible]

Esto quiere decir que excepto esas cosas raras que están ahí, todo lo demás son palabras que tienen su respectivo tag. Concluimos entonces que para filtrar las palabras validas deberíamos buscar que esté en alguno de los dos conjuntos *spelling* o *lexicon* y, sí es así, lo se puede etiquetar con uno de los dos sistemas:

```

from pattern.es import lexicon, spelling, tag

def clasificar(palabra):
    print(tag(palabra, tokenize=True, encoding='utf-8', tagset =
        ↪ 'UNIVERSAL'))
    print(tag(palabra, tokenize=True, encoding='utf-8'))

palabra = 'azucar'
if not palabra in spelling:
    if not palabra in lexicon:
        print('No se encuentra en pattern.es')
    else:
        print('La encontré en lexicon')
        clasificar(palabra)
else:
    print('La encontré en spelling')
    clasificar(palabra)

```

Probamos distintas palabras:

A. Etiquetado en Pattern

```
Camino
La encontró en lexicon
[('Camino', 'NCS')]
[('Camino', 'NN')]

camion
No se encuentra en pattern.es

cami  n
La encontr   en lexicon
[('cami  n', 'NCS')]
[('cami  n', 'NN')]

vurro
No se encuentra en pattern.es

burro
La encontr   en spelling
[('burro', 'NCS')]
[('burro', 'NN')]

Burro
No se encuentra en pattern.es

BURRO
No se encuentra en pattern.es

Argentina
La encontr   en lexicon
[('Argentina', 'NP')]
[('Argentina', 'NNP')]

argentina
La encontr   en spelling
[('argentina', 'AQ')]
[('argentina', 'JJ')]

ARGENTINA
No se encuentra en pattern.es
```

Falta perfeccionarlo, ya que algunos t  rminos espec  ficos no los encuentra y hay que filtrar las may  sculas, no todas, ya que los **sustantivos propios** en lexicon est  n con may  scula.

Además existe otra lista de palabras: `verbs` que tiene pocas palabras en relación a los otros, con lo cual sirve para buscar en ella primero que es más rápido. Sumando las funciones para capitalizar del modulo `string`:

```
from pattern.es import verbs, tag, spelling, lexicon
import string
def clasificar(palabra):
    print( tag(palabra, tokenize=True, encoding='utf-8',tagset =
    ↪ 'UNIVERSAL'))
    print( tag(palabra, tokenize=True, encoding='utf-8'))
    print()

palabra = 'Camino'
print(palabra)
while palabra != 'q':
    if not palabra.lower() in verbs:
        if not palabra.lower() in spelling:
            if (not(palabra.lower() in lexicon) and
            ↪ not(palabra.upper() in lexicon) and
            ↪ not(palabra.capitalize() in lexicon)):
                print('No se encuentra en pattern.es')
            else:
                print('La encontré en lexicon')
                clasificar(palabra)
        else:
            print('La encontré en spelling')
            clasificar(palabra)
    else:
        print('La encontré en verbs')
        clasificar(palabra)

    print()
    palabra = input()
```

Algunos ejemplos:

A. Etiquetado en Pattern

```
BURRO
La encontró en spelling
[('BURRO', 'NCS')]
[('BURRO', 'NN')]

ArMoNíA
La encontró en lexicon
[('ArMoNíA', 'NCS')]
[('ArMoNíA', 'NN')]

BaILAr
La encontró en verbs
[('BaILAr', 'VMN')]
[('BaILAr', 'VB')]

djjd
No se encuentra en pattern.es
```


B. Raspberry Py

B.1. Raspeberry Pi

B.1.1. Primer aproximación

- Simulador web de Sensor HAT (ya tiene led y sensores integrados):
<https://trinket.io/library/trinkets/5b83aa39e6>
Es mas simple pero para probar algunos comandos está piola. La desventaja es que no se puede instalar cualquier librería. Sirve como una primera aproximación a una Raspberry Pi.

B.1.2. Probando el sistema Raspbian

Para simular la **Raspberry** se puede usar [VirtualBox](#) e instalar una maquina con [Raspbian](#), o en Windows directamente usar [Qemu](#). Para mi anda mejor VirtualBox porque le puedes asignar más memoria ram al emulador. Mas informacion de como instalar la máquina virtual [aqui](#)

Para tener copy paste entre el sistema anfitrión y la maquina virtual, insertar el cd de adicionales desde el menu *Devices* y luego en una terminal:

```
$ sh /media/cdrom/VBoxLinuxAdditions.run
```

B.1.3. Instalación de requisitos

- [RPi.GPIO Installation](#)

```
$ sudo apt-get update  
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```
- [Adafruit_Python_DHT Installation](#)

```
$ sudo apt-get install python-pip  
$ sudo python -m pip install --upgrade pip setuptools wheel  
$ sudo pip install Adafruit_DHT
```
- [Luma.LED_Matrix: Display drivers for MAX7219, WS2812, APA102](#)

B. Raspberry Py

```
$ sudo usermod -a -G spi,gpio pi
$ sudo apt-get install build-essential python-dev python-pip libfreetype6-dev
$ sudo -H pip install --upgrade --ignore-installed pip setuptools
$ sudo -H pip install --upgrade luma.led_matrix
```

- [Luma.Examples](#)

```
$ sudo usermod -a -G i2c,spi,gpio pi
$ sudo apt install python-dev python-pip libfreetype6-dev libjpeg-dev build-es
$ sudo apt install libsdl-dev libportmidi-dev libsdl-ttf2.0-dev libsdl-mixer1
```

- Log out y volver a entrar. clonar repo:

```
$ git clone https://github.com/rm-hull/luma.examples.git
$ cd luma.examples
```

- Instalar las librerías:

```
$ sudo -H pip install -e .
```

- Correr ejemplos:

```
$ python luma.examples/examples/3d_box.py
```

(esto último solo andará en una verdadera Raspberry Pi, ya que hace uso de los sensores y dispositivos que el emulador no posee *a priori*. Para ello dirigirse al **final** de este documento.)

B.1.4. Usando los sensores y dispositivos

Trabajar con GPIO (General Purpose I/O, los pines de la placa):

- Pinout: <https://github.com/splitbrain/rpiplusleaf>



<https://user-images.githubusercontent.com/11953173/59568951-fa28a400-9058-11e9-8000-000000000000>

- Interactivo: https://pinout.xyz/pinout/pin33_gpio13

- Uso Basico (esto lo vamos a usar para el **microfono**):

<https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/>
<https://www.raspberrypi.org/documentation/usage/gpio/>

“pyhton

import RPi.GPIO as GPIO

```
GPIO.setmode(GPIO.BOARD)
# or
GPIO.setmode(GPIO.BCM)

GPIO.setup(channel, GPIO.IN)

# Read
GPIO.input(channel)

# Set
GPIO.output(channel, state)

# Poll
if GPIO.input(channel):
    print('Input was HIGH')
else:
    print('Input was LOW')

# To wait for a button press by polling in a loop:
while GPIO.input(channel) == GPIO.LOW:
    time.sleep(0.01) # wait 10 ms to give CPU chance to do other things

# To clean up at the end of your script:
GPIO.cleanup()
““
```

Para el sensor de temp y humedad:



[https://github.com/adafruit/Adafruit_Python_DHT/blob/master/examples/simpletest.](https://github.com/adafruit/Adafruit_Python_DHT/blob/master/examples/simpletest.py)

[py](https://github.com/adafruit/Adafruit_Python_DHT/blob/master/examples/simpletest.py)

```
“python
import Adafruit_DHT
sensor = Adafruit_DHT.DHT22
pin = 23

hum, temp = Adafruit_DHT.read_retry(sensor, pin)
““
```

Para mostrar la info en las matrices de LED:

<https://luma-led-matrix.readthedocs.io/en/latest/python-usage.html#x8-led-matrices>
<https://github.com/rm-hull/luma.examples>

```
“python
from luma.core.interface.serial import spi, noop
from luma.led_matrix.device import max7219
from luma.core.render import canvas
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT, SINCLAIR_FONT,
LCD_FONT
from luma.core.virtual import viewport

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=2, block_orientation=-90)
device.contrast(0x05)
msg = 'asd'
# para mostrar un mensaje que vaya pasando usar show_message:
show_message(device, msg, fill="white", font=proportional(LCD_FONT), scroll_delay=0.05)

# para mostrar un mensaje estático usar draw y time para ir actualizándolo:
while True: # o import repeat y repeat(None).
time.sleep(1)
msg = time.asctime()
msg= time.strftime("%H %M")
with canvas(device) as draw:
text(draw, (1, 0), msg, fill="white")
time.sleep(2)
pass # ???
“
```

- para dibujar en las celdas y obtener el código:
<http://dotmatrixtool.com/>

Para emular:

Primero instalamos [Luma.emulation](#)

```
$ sudo apt install python-dev python-pip build-essential
$ sudo apt install libsdl-dev libportmidi-dev libsdl-ttf2.0-dev libsdl-mixer1.2-dev
$ sudo pip install --upgrade luma.emulator
```

(recordar hacer lo mismo con **python3** y **pip3**)

Hay que correr los archivos con unos parámetros especiales. Los más importantes son:

```
--display DISPLAY, -d DISPLAY
    Display type, supports real devices or emulators.
    Allowed values are: ssd1306, ssd1309, ssd1322,
    ssd1325, ssd1327, ssd1331, ssd1351, sh1106, pcd8544,
    st7735, ht1621, uc1701x, st7567, max7219, ws2812,
    neopixel, neosegment, apa102, capture, gifanim,
    pygame, asciiart, asciiblock (default: ssd1306)
```

De estas opciones notar:

- max7219: Este es cuando tengamos el dispositivo real.
- capture: Este saca una instantanea de lo que mostraria el display y lo guarda como png.
- gifanim: Este es como capture pero guarda un gif animado.
- **pygame**: Este muestra el output en una ventana en tiempo real

Seguimos con los parametros restantes

```
--width WIDTH          Width of the device in pixels (default: 128)

--height HEIGHT        Height of the device in pixels (default: 64)

--rotate ROTATION, -r ROTATION
    Rotation factor. Allowed values are: 0, 1, 2, 3
    (default: 0)

--transform TRANSFORM
    Scaling transform to apply (emulator only). Allowed
    values are: identity, led_matrix, none, scale2x,
    seven_segment, smoothscale (default: scale2x)

--scale SCALE          Scaling factor to apply (emulator only) (default: 2)
```

B.1.5. Por ejemplo, para simular dos modulos de 8x8 en matriz de led en tiempo real:

El codigo hay que cambiarlo un poco:

```
[] from luma.core.interface.serial import spi, noop from luma.led_matrix.device import max7219 from luma.core.legacy
comentar las lineas: serial = spi(port=0, device=0, gpio=noop()) device = max7219(serial,
cascaded=2, block_orientation = -90)
```

B. Raspberry Py

```
device = get_device()
```

```
msg = 'Aguante Python y la UNLP 2k19' show_message(device, msg, fill = "white", font = proportion)
```

Y ejecutarlo con los parámetros:

```
$ python luma.examples/examples/archivito.py --display pygame --transform  
led_matrix --width 16 --height 8
```

| *Nota:* ya que hay que usar las librerías demo lo mas facil es meter el archivo en la
carpeta *luma.examples/examples* que se creó cuando clonamos el git |

| — |