Trabajo Práctico Final, Progr. 2

Ivan Campos Wainer FCEIA, UNR

Resumen del objetivo

> Proceso de selección de conceptos

Antes de seleccionar mi estrategia final para abarcar este trabajo, lleve adelante un proceso de búsqueda de distintas formas para alcanzar el objetivo

Primer concepto

A partir de la obtención del archivo y su posterior separación a través de un split, conseguí una lista con las palabras del mismo. A partir de esto, llevé a cabo la creación de un diccionario en el que las claves eran cada palabra del archivo, y su valor, una lista que contiene las palabras que la anteceden y preceden (teniendo en cuenta que puede ser mas de una opción).

Por ejemplo, la frase “**el** amor verdadero y **el** agua es azul”

```
{
  "el": [“amor”, “y”, “agua”]
}
```

La gran desventaja que encontré, es que al usar un diccionario de N claves (N = cantidad de palabras), y una lista compuesta por valores de uno como longitud mínima, se vuelve compleja de recorrer y quedan M valores (en el mejor de los casos, M=N) que se vuelven innecesarios.

Segundo Concepto

Buscando otra alternativa, intenté asignar una probabilidad a cada palabra del texto.

Para llevarlo a cabo, guardé en un diccionario cada palabra como una clave, pero asignándole un porcentaje como valor, y tomando como probabilidad la cantidad de

veces que la misma aparecía dentro del texto.

Por ejemplo, así quedaría la siguiente oración “el amor despues del amor”:

```
{
  "el": 20,
  “amor”: 40,
  “despues”: 20,
  “del”: 20
}
```

A diferencia de las ideas anteriores, esta posibilidad tiene como valor un número (float) y no posee una lista de N elementos. Sin embargo, al igual que la idea inicial, tiene como desventaja que debe guardar todas las palabras del texto dentro de una estructura, implicando elementos no deseados.

Finalmente, como ninguna de estas ideas me pareció totalmente adecuada para solucionar la consigna, planteé la siguiente idea.

Concepto elegido

Por últmo, separé las palabras que estaban alrededor del guión bajo de la frase y busqué en la letra de la canción los siguientes puntos para guardarlas en un diccionario junto a la probabilidad de aparición:

- La palabra que la precede, en caso de que el guión se encuentre detrás
- La palabra que la antecede, en caso de que el guión se encuentre delante

Por ejemplo:

Letra:

> El amor, después del amor tal vez
>
> Se parezca a este rayo de sol
>
> Y ahora que busque y ahora que encontre
>
> El perfume que lleva el dolor

Frase:

> Y ahora que _ y ahora que encontre

Diccionario:

```
{
  "busque": 2,
  "encontre": 1
}
```

Resumen parseo de texto en C:

Para el parseo en C, lo primero que hice fue crear el archivo de texto que tiene en su interior los nombres de todos los archivos contenidos en la carpeta del artista separadas por un '\n'. Luego, por cada una de las canciones, apliqué la función add_text_to_output que, como explico en el código, se encarga de iterar por cada caracter contenido en el archivo y "empujarlos" a uno nuevo, formateando el texto. Para finalizar, el programa ejecuta el codigo en Python.

Resumen sustitución de barra baja y archivo de salida en Python:

Primero, el programa lee el archivo del texto formateado, la letra del artista y las frases. A continuación, formatea las palabras que incluye cada una, actuando como un auxiliar que sustituye las mayúsculas y elimina los puntos y comas de las oraciones. Después, separa las palabras que rodean al guión y llama a la función más importante del programa: add_words que verifica 5 casos:

- Cuando el guión bajo se encuentra entre las dos palabras coincidentes

  Como vemos en el ejemplo, el guión bajo se encuentra entre dos palabras presentes en la canción: "que" e "y". Y es por eso que se agrega al diccionario con +100 de chances de ser el que sustituya el mismo.

- Cuando el guión bajo está primero en la frase

  Se puede dar que el guión se encuentre al principio de la frase, por lo que buscará unicamente en la letra aquellas palabras que están detrás de la que precede el guión.

- Cuando el guión bajo esta a lo último de la frase

  Se puede dar que el guión se encuentre al final de la frase, por lo que buscará unicamente en la letra aquellas palabras que están delante de la que antecede el guión.

- Cuando la palabra está atrás o adelante del guión

  El diccionario contiene las palabras y su posibilidad de sustituir el guión bajo con la que se encuentre adelante y atrás del guión mientra recorre el texto.
  Aclaración: Es importante que este caso se verifique luego de los desarrollados anteriormente.

- Cuando se necesite aumentar el radio

  En algunos casos, las palabras que rodean el guión, no se encuentran en el texto. Por lo tanto, se debe aumenta el radio y se busca en i elementos detras y delante del mismo, siendo i = palabras distancia del guión.

Fuera de add_words, en caso de que ninguna palabra que rodee el guión bajo exista dentro de la letra de la canción, se le asigna una palabra random que se encuentre en ella.

Para finalizar, busca las palabras con las probabilidades más altas dentro del diccionario y sustituye la barra baja con las mismas.

Serie de pasos

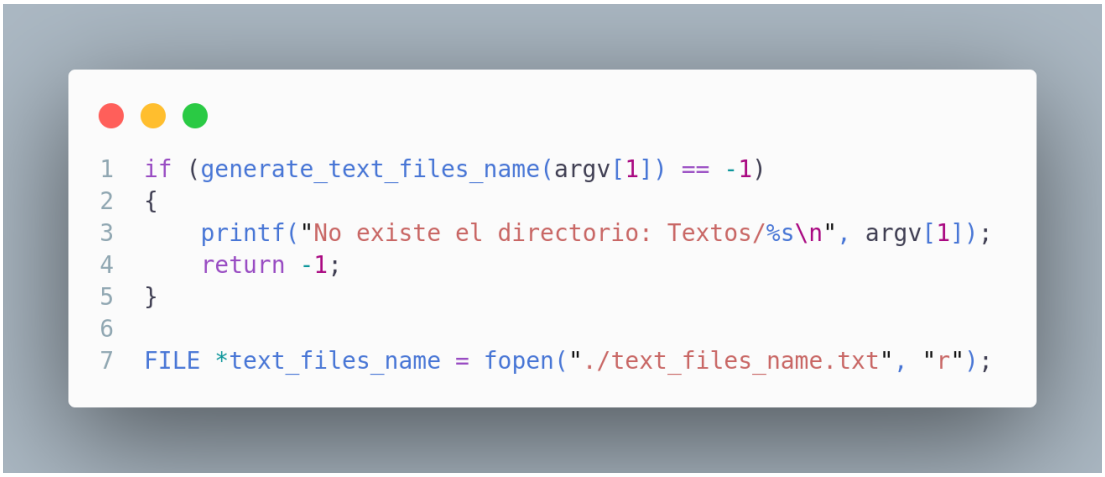
> A continuación, desarrollé la serie de pasos que lleve adelante para hacer este trabajo

▼ Parseo de texto en C

1. Ejecución de generate_text_files_name y apertura de archivo

```c
/* Recibe como parametro el nombre del directorio al que debemos acceder
dentro de Textos/ luego, ejecuta el comando ls y guarda en el directorio raíz, un archivo llamado
text_files_name.txt */
int generate_text_files_name(char name[])
{
    char command[100];

    sprintf(command, "./Textos/%s", name);
    FILE *faux = fopen(command, "r");

    if (!faux)
    {
        return -1;
    }

    fclose(faux);

    sprintf(command, "cd Textos/%s && ls > ../../text_files_name.txt", name);
    system(command);

    return 1;
}
```

Definición de la función



```c
if (generate_text_files_name(argv[1]) == -1)
{
    printf("No existe el directorio: Textos/%s\n", argv[1]);
    return -1;
}

FILE *text_files_name = fopen("./text_files_name.txt", "r");
```

Llamada de la función y apertura del archivo en main

```
cantarhastamorir.txt
coloresperanza.txt
guapa.txt
quenomepierda.txt
tratardeestarmejor.txt
```

Ejemplo de salida de text_files_name.txt

2. Declaración de string auxiliar path y creación de archivo de salida, (Entradas/nombre.txt)

```c
char path[100]; // String auxiliar para guardar rutas (se irá modificando a lo largo del programa)
sprintf(path, "./Entradas/%s.txt", argv[1]);
FILE *file_output = fopen(path, "w");
```

Aclaración: se abre en modo ‘w’ para sobrescribir el archivo del artista en caso de que se ejecute de nuevo el programa.

3. Declaración del bucle while por cada línea de text_files_name y ejecución de add_text_to_output por cada archivo recorrido (almacenado en path)

```c
while ((fscanf(text_files_name, "%s", path)) != EOF)
{
    add_text_to_output(argv[1], path, file_output); // <--
}
```

```c
void add_text_to_output(char *name, char *filetxt, FILE *file_output)
{
    // ...
}

```

Le pasa por parámetros el nombre del artista, el path, que sería el nombre del archivo recorrido y el file_output (Entradas/)

- a. Declaración de variable auxiliar path y archivos

```c
char path[100];
sprintf(path, "./Textos/%s/%s", name, filetxt);
FILE *file_input = fopen(path, "r");
```

- b. Declaración de ciclo while y variables auxiliares character, prev_character

```c
char character, prev_character;
while ((character = fgetc(file_input)) != EOF)
{
    //...
}
```

Al igual que antes, recorre el archivo de la letra (file_input)

- c. Condiciones para el parseo (para poner carácter en el file_output) y sustitución de caracteres

```c
while ((character = fgetc(file_input)) != EOF)
{
    if (esMayus(character))
        fputc(character + 32, file_output);
    else if (character == '\n' && prev_character != '.')
        fputc(' ', file_output);
    else if (character != '.' && character != ',')
    {
        fputc(character, file_output);
    }

    prev_character = character;
}
```

- d. Agregado de \n para evitar bugs y cierre de archivo

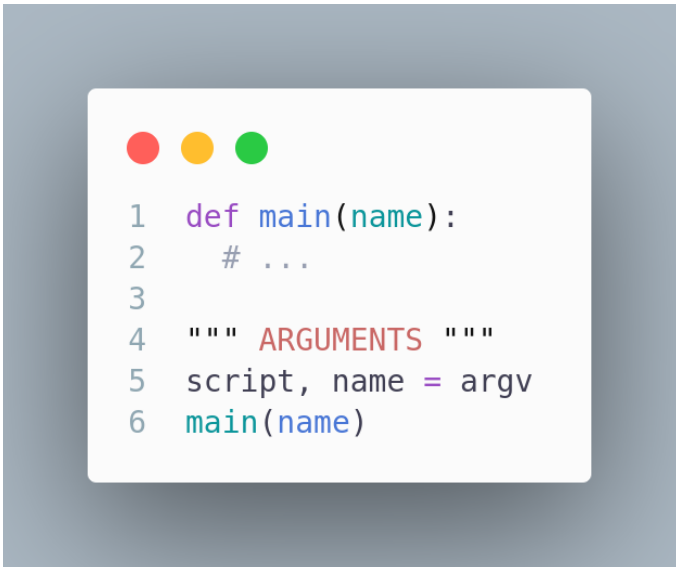
```c
/* d. Agregado de \n para evitar bugs y cierre de archivo */
fputc('\n', file_output);

fclose(file_input);
```

4. Liberación de memoria, cierre de archivos y llamada al programa Python

```c
fclose(file_output);
fclose(text_files_name);

sprintf(path, "python3 main.py %s", argv[1]);
system(path);
```

▼ Sustitución de barra baja y archivo de salida en Python

1. Declaración de la función main y argumentos



```python
def main(name):
  # ...

""" ARGUMENTS """
script, name = argv
main(name)
```

2. Apertura de archivo de los textos parseados; creación de lista

```python
f_lyrics = open("./Entradas/" + name + ".txt")
lyrics = f_lyrics.read().replace("\n", " ").split(" ")  # La lista contiene como elementos cada palabra del archivo
f_lyrics.close()

```

La lista contiene como elementos cada palabra del archivo

3. Declaración de f_phrase y ejecución de read_file (función auxiliar) y creación de archivo de salida

```python
f_phrase = read_file('./Frases/' + name + '.txt')

file_output = open("./Salidas/" + name + ".txt", 'w+')
```

Llamada a la función desde el main

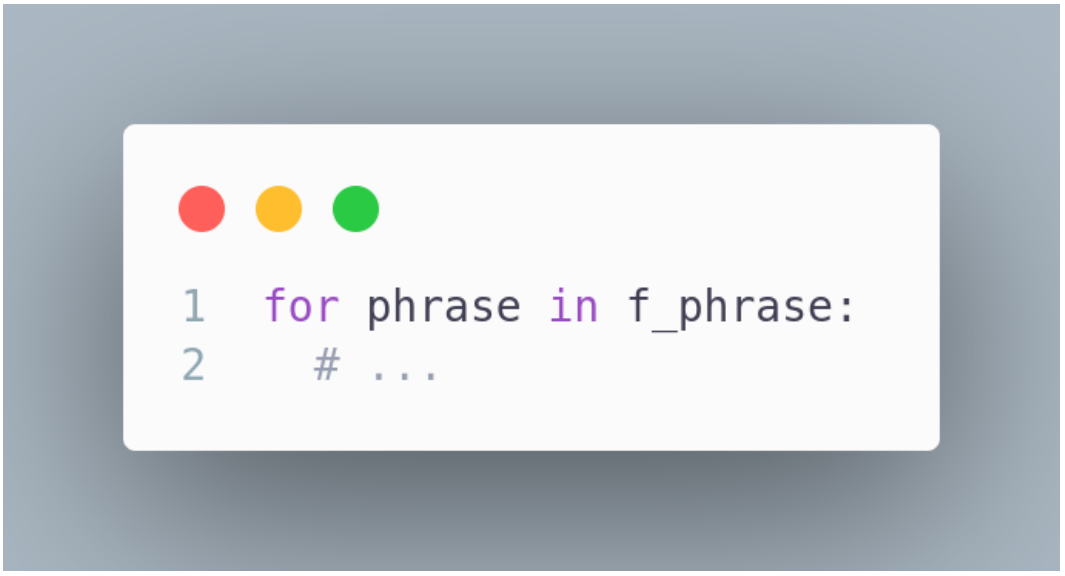
```python
""" Se encarga de obtener el archivo que contiene las frases del artista
para luego iterar en cada salto de linea:
  - guardar su contenido en forma de lista (word_line_aux)
  - eliminar las comillas simples '' que quedan de sobra en word_line_aux
  - guarda word_line_aux en content
por ultimo, reotrna content, que es una lista que contiene listas de cada palabra contenida en linea
del archivo frase """
def read_file(file_name):
  file = open(file_name, 'r')
  lines_of_text = file.readlines()

  content = []

  for line_of_text in lines_of_text:
    word_line_aux = line_of_text.replace('\n', '').split(' ')

    # Eliminar '' de la lista
    if '' in word_line_aux:
      word_line_aux.remove('')

    content.append(word_line_aux) # Guarda ['Te', '_,', 'unos', 'chinos', 'en', 'madrid'] dentro de la lista

  file.close()

  return content
```

Función auxiliar read_file

```
""" Frases/Fito_Paez.txt """
Te _, fumabas unos chinos en madrid
nadie puede y nadie debe vivir _ amor

f_phrase = [
    ['Te', '_,', 'fumabas', 'unos', 'chinos', 'en', 'madrid'],
    ['nadie', 'puede', 'y', 'nadie', 'debe', 'vivir', '_', 'amor']
]
```

Ejemplo de cual sería el resultado de f_phrase con ese archivo .txt

4. Iteración por cada frase en lista de frases

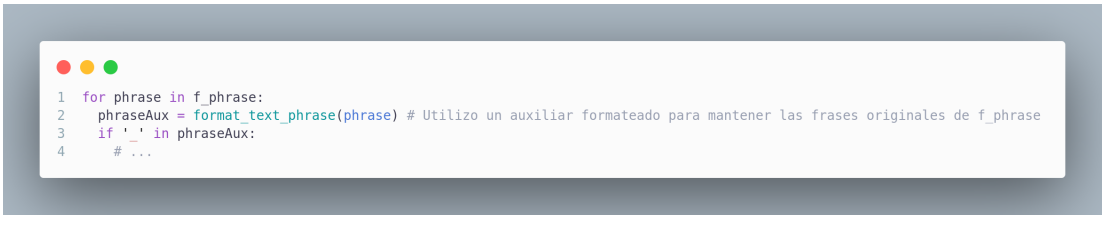


Recordemos que f_phrase es una lista de listas. Entonces phrase representa una lista de una frase

5. Declaración de phraseAux (frase formateada para trabajar con ella), ejecución de format_text_phrase

```
"""
format_text_phrase: List(String) -> None | List(String)
format_text_phrase se encarga de formatear la frase, eliminado comas, puntos
después e independientemente de '_'. También, convierte las mayusculas en minusculas.
"""
def format_text_phrase(phrase): # Retorna NONE en caso de que no haya '_'
  phraseAux = ' '.join(phrase).replace("_,", "_").replace("_.", "_").replace(".", "").lower().split(" ")
  return phraseAux

```

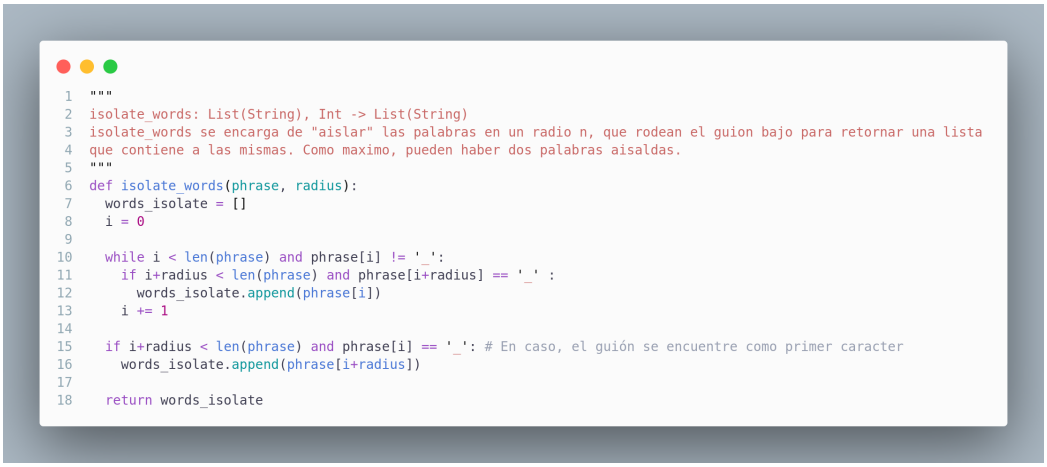
Definición de la función



```
for phrase in f_phrase:
  phraseAux = format_text_phrase(phrase) # Utilizo un auxiliar formateado para mantener las frases originales de f_phrase
  if '_' in phraseAux:
    # ...
```

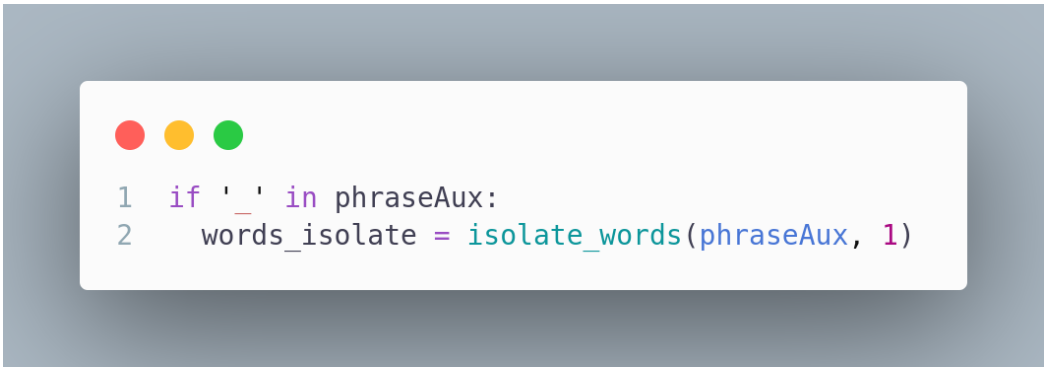
Verifica si existe ‘_’, debido a que puede haber frases que no contengan _

- a. Declaración de words_isolate y ejecución de isolate_words



```
"""
isolate_words: List(String), Int -> List(String)
isolate_words se encarga de "aislar" las palabras en un radio n, que rodean el guion bajo para retornar una lista
que contiene a las mismas. Como maximo, pueden haber dos palabras aisaldas.
"""
def isolate_words(phrase, radius):
  words_isolate = []
  i = 0

  while i < len(phrase) and phrase[i] != '_':
    if i+radius < len(phrase) and phrase[i+radius] == '_' :
      words_isolate.append(phrase[i])
    i += 1

  if i+radius < len(phrase) and phrase[i] == '_': # En caso, el guión se encuentre como primer caracter
    words_isolate.append(phrase[i+radius])

  return words_isolate
```

Definición de la función isolate_words



```
if '_' in phraseAux:
  words_isolate = isolate_words(phraseAux, 1)
```

Devuelve una lista como máximo de dos elementos, que contiene las words que rodean el guion bajo

- b. Declaración de dict (diccionario con las palabras y probabilidad de que sustituya el '_') y ejecución add_words

```
""" add_words: List(String), List(String), List(String) -> dict
add_words recibe como parametro la letra, la lista de las palabras aisladas que
rodean el '_' (retorna isolate_words) y la lista que contiene las frases formateadas.
Sus varios ifs sirven para verificar casos unicos en el que el programa sufría un bug. Como
cuando el '_' es el primer elemento, o el ultimo; también cuando se necesita aumentar el radio.

Retorna un diccionario, sus claves serán las posibles palabras encontradas y sus valores sus posibilidades.
Aclaracion: En caso de estar entre dos palabras conocidas en el texto, se le suma en vez de una, dos chances, debido
a que es mas probable"""
def add_words(lyrics, words_isolate, f_phrase):
  dict = {}

  for i in range(0, len(lyrics)):
    # Caso en el que el Guion baje se encuentre entre las dos palabras coincidentes
    if len(words_isolate) == 2 and i+2 < len(lyrics)-1 and lyrics[i] == words_isolate[0] and lyrics[i+2] == words_isolate[1]:
      dict[lyrics[i+1]] = dict.get(lyrics[i+1], 0) + 2
    elif len(words_isolate) == 2 and i-2 > 0 and lyrics[i] == words_isolate[1] and lyrics[i-2] == words_isolate[0]:
      dict[lyrics[i-1]] = dict.get(lyrics[i-1], 0) + 2

    # Caso en que el Guion bajo esté primero
    elif f_phrase[0] == '_' and lyrics[i] == words_isolate[0]:
      dict[lyrics[i-1]] = dict.get(lyrics[i-1], 0) + 1
    # Caso en que el Guion bajo esté ultimo
    elif f_phrase[-1] == '_' and lyrics[i] == words_isolate[0]:
      dict[lyrics[i+1]] = dict.get(lyrics[i+1], 0) + 1

    # Aumenta el indice de la palabra (key del dict) que se encuentra detras el guión bajo
    elif len(words_isolate) == 2 and lyrics[i] == words_isolate[0]:
      dict[lyrics[i+1]] = dict.get(lyrics[i+1], 0) + 1
    # Aumenta el indice de la palabra (key del dict) que se encuentra despues el guión bajo
    elif len(words_isolate) == 2 and lyrics[i] == words_isolate[1]:
      dict[lyrics[i-1]] = dict.get(lyrics[i-1], 0) + 1

    elif(lyrics[i] == words_isolate[0]): # Caso de que se encuentre en un radio mayor.
      dict[lyrics[i+1]] = dict.get(lyrics[i+1], 0) + 1
      dict[lyrics[i-1]] = dict.get(lyrics[i-1], 0) + 1

  return dict
```

```
if '_' in phraseAux:
  words_isolate = isolate_words(phraseAux, 1)

  dict = add_words(lyrics, words_isolate, phraseAux) # <--
```

Agrega palabras que se encuentran entre los dos elementos, o el elemento, en caso de que el guion bajo se encuentre ultimo o primero

- c. Caso de que las palabras que rodean el guion bajo, no existan, aumenta el radio

```
i = 2
while i < len(phrase)+1 and len(dict) == 0:
  try:
    dict = add_words(lyrics, isolate_words(phraseAux, i), phraseAux)
    i+=1
  except:
    dict[lyrics[randint(0, len(lyrics))]]
```

Se pueden dar varios casos, en el que la palabra que antecede o precede el guion, no existan en la letra. Por lo tanto, se aumenta el radio y se busca la palabra que está detrás de la palabra que antecede el '_'; el radio se puede aumentar N cantidad de veces (siendo N=longitud de la frase + 1). Es importante aclarar, que se busca generar una excepción en caso de que ninguna de las palabras que están en la frase existan en la letra; por lo tanto, se le asignará una palabra random (debido a que tiene la misma probabilidad de sustituir el '_' como cualquier otra)

- c. Llamada a la función print_phrase_in_output_file encargada de "colocar" frase en archivo de salida con sus correspondiente guion bajo sustituido

```
"""
print_phrase_in_output_file: List(String), dict, FILE -> None
Recibe una lista de la frase sin modificar, el diccionario con la clave de las palabras mas
probables y su valor de probabilidad y el archivo de salida donde serán escrita la frase.
Crea la variable keys_max_val, que obtiene una lista con las CLAVES del diccionario con mayores valores.
Luego, obtiene una clave random de la lista keys_max_val; esto se debe a que si hay palabras que tienen
el mayor valor del diccionario, eligirá una random porque tiene la misma probabilidad de ser la que sustituya
el guion bajo.
Finalmente, sustituye el '_' con la palabra seleccionada, y la escribe en el archivo de salida
"""
def print_phrase_in_output_file(phrase, dict, file_output):
  keys_max_val = get_keys_max_val(dict)
  random_word = keys_max_val[randint(0, len(keys_max_val) - 1)] # Caso en el que las probabilidades de las palabras en el dicionario sean iguales
  final_phrase = ' '.join(phrase).replace('_', random_word)

  file_output.write(final_phrase)
  file_output.write('\n')
```

Definición de la función print_phrase_in_output_file

```
print_phrase_in_output_file(phrase, dict, file_output)
```

Llamada de la función dentro del for (fuera del if)