


Open · Opened 4 days ago by  **Rubén Montero**

Intentemos que sea razonable

Resumen

- Intentaremos que nuestro **ChatBot** sea lo *más* razonable posible

Descripción

Nuestro **ChatBot** sólo responde ¡**Vaya!** a todo lo que le decimos. No parece muy inteligente.

Aunque las [IA conversacionales](#) son todo un mundo, nosotros vamos a implementar una versión *sencilla* que no es inteligente realmente, pero lo intenta.

¿En qué consiste?

Cuando leamos la *entrada de usuario*, vamos a buscar en su *contenido*.

Si aparece *alguna* palabra conocida, responderemos con una frase *genérica* asociada a la palabra.

¿Frases conocidas?

Sí, tendremos esta información almacenada por nuestro **ChatBot** :

palabra	frase para responder
amo	¿No crees que el amor y el odio están separados por una delgada línea?
apagar	Apagar, encender,... Qué más da
casa	Para mí una casa es un montón de circuitos
color	Ojalá pudiera ver los colores como tú
dinero	Si hay algo que sé es que el dinero no da la felicidad. Y no sé mucho...
favor	Yo no hago favores
humano	Los humanos siempre andáis a lo vuestro
inteligencia	¿Inteligencia? No me hables de inteligencia...
interesante	Para cosas interesantes, el Discovery Channel
máquina	¿Has visto la película Terminator? Mejor hazlo...
ordenador	Los ordenadores son máquinas muy útiles
programa	Tu lavadora también tiene programas, ¿sabías?
quiero	Querer es una palabra con un significado muy amplio
(sin coincidencias)	Vaya...

¿Y esto será suficiente?

No se asemeja ni por asomo a una IA real, pero puede dar algo de juego.

¿Cómo lo implementamos?

Usaremos un *atributo* de tipo *diccionario*.

En Python se declaran con *llaves* (`{ }`) y funcionan igual que un JSON.

Declara un atributo que contenga la información de la tabla como **clave-valor** en el **constructor**:

```
class ChatBot:
    # ...

    def __init__(self):
        # (...) Tareas anteriores
        # (...) Tareas anteriores
        self.knowledge = {
            "amo": "¿No crees que el amor y el odio están separados por una delgada línea?",
            "apagar": "Apagar, encender,... Qué más da",
            "casa": "Para mí una casa es un montón de circuitos"
            # El resto de valores
            # ...

            # NO hace falta añadir el valor 'sin coincidencias'
        }
```

Nuestro ChatBot ya tiene knowledge

Ahora **usémosla**.

Existen muchas formas de desempeñar esta tarea, y muchas más *eficientes* que lo que vamos a hacer ([búsqueda binaria](#), separación en palabras y acceso directo...)

Nosotros nos limitaremos a **recorrer** el diccionario.

¿Se pueden recorrer diccionarios en Python?

Sí.

¿Igual que listas?

Algo parecido. Se recorren las **claves**.

```
>>> a_dict = {'color': 'azul', 'fruta': 'manzana', 'mascota': 'perro'}
>>> for key in a_dict:
...     print(key)
...
color
fruta
mascota
```

Y como bien sabes, con una **clave** podemos acceder a su **valor** asociado usando *corchetes* (`[]`):

```
>>> a_dict = {'color': 'azul', 'fruta': 'manzana', 'mascota': 'perro'}
>>> for key in a_dict:
...     print(a_dict[key])
...
azul
manzana
perro
```

¿Y nosotros qué hacemos?

Añadamos un nuevo método para *encapsular* la complejidad de esto, y no **enmarañar** demasiado `begin_conversation` :

```
def begin_conversation(self):
    print(self.name + " dice:")
    print("¡Hola! Soy " + self.name + ". ¿De qué quieres hablar?")
    print("(Cuando quieras despedirte, di 'salir')")
    print("")
    user_input = input("Tú dices: ")
    while user_input != "salir":
        print("")
        print(self.name + " dice:")
        - print("¡Vaya!")
        + print(self.__response_for(user_input))
        print("")
        user_input = input("Tú dices: ")
    print("")
```

```

        print(self.name + " dice:")
        print("¡Hasta pronto!")
+
+     def __response_for(self, text):
+         # Aquí devolveremos la respuesta apropiada

```

¿Y esas dobles *barras bajas*?

Te has fijado que `__response_for` empieza por unos caracteres extraños.

Esta convención de nombrado Python sirve para indicar que el método es *privado* y *no debe ser invocado* fuera de la clase.

Como sabes, todo es **público** en Python. Pero con esta convención, al menos podemos indicar que *no esperamos que el método se use desde fuera* (aunque se podría).

`__response_for`

Recibimos por **parámetro** la entrada de usuario:

```

def __response_for(self, text):
    # Si el usuario ha tecleado 'Te quiero',
    # entonces text='Te quiero'

```

Recorremos el diccionario con el conocimiento de nuestro **ChatBot**:

```

def __response_for(self, text):
    for word in self.knowledge:
        # Por cada clave del diccionario...

```

Y usamos `__contains__`, que sirve para verificar si *un string* está contenido en otro:

```

def __response_for(self, text):
    for word in self.knowledge:
        if text.__contains__(word):
            # El texto contiene una palabra conocida

```

Si es así, ¡qué bien!

Podemos devolver la respuesta apropiada (el **valor** asociado a la **clave**):

```

def __response_for(self, text):
    for word in self.knowledge:
        if text.__contains__(word):
            return self.knowledge[word]

```

¿Y si no?

Si la ejecución del bucle termina y **no** ha habido éxito, entonces ningún `return` ha tenido lugar.

Haremos un `return` por defecto:

```

def __response_for(self, text):
    for word in self.knowledge:
        if text.__contains__(word):
            return self.knowledge[word]
    return "Vaya..."

```

¡Listo!

Ya puedes disfrutar de horas y horas de conversación con tu **ChatBot**

Por último

Verifica que tu código pasa el *test* asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.



Rubén Montero @ruben.montero changed milestone to [%Sprint 3](#) 4 days ago