Categorizacion de Informacion en Redes Sociales. Aplicacion a la Ciberseguridad.

API Documentation

June 10, 2017

Contents

1 Module alertaUsuario

Este módulo contiene la clase **AlertaUsuario**.

Author: Rubén Jiménez Ortega.

Version: 1.0

1.1 Class AlertaUsuario

??-1

alertaUsuario.AlertaUsuario

Crea un QMessageBox (es un diálogo utilizado para mostrar que el nodo Raíz a sido cambiado con éxito).

1.1.1 Methods

___init___(*self*)

Constructor por defecto de la clase **AlertaUsuario**.

2 Module categoria

Este módulo contiene la clase **Categoria**.

Author: Rubén Jiménez Ortega.

Version: 1.0

2.1 Class Categoria

??-2

categoria.Categoria

Crea un QWidget para mostrar los tweets relacionados con una categoría en concreto.

2.1.1 Methods

___init___(*self*, *titulo*, *categoria*, *contador*, *tweets*)

Constructor con parámetros de la clase **Categoria**.

Parameters

`titulo:` Variable para cambiar el título de la nueva ventana.
(type=str)

`categoria:` Variable que almacena la categoría que mostramos.
(type=str)

`contador:` Número de tweets de la categoría que estudiamos.
(type=int)

`tweets:` Tweets del usuario y de la categoría elegida almacenados en una lista para poder mostrarlos.
(type=list)

center(*self*)

Función para centrar la ventana categoría en una pantalla, independientemente del tamaño de la misma.

3 Module creaVentanas

Este módulo contiene la clase **CreaVentanas**.

Author: Rubén Jiménez Ortega.

Version: 1.0

3.1 Class CreaVentanas

Clase para crear la mayoría de las ventanas de las que dispone la aplicación.

3.1.1 Methods

center(*self*, *qtgui*)

Función para centrar cualquier ventana en una pantalla, independientemente del tamaño de la misma.

Parameters

`qtgui:` Ventana que queremos centrar.

crearVentanaInformacion(*self*, *nombreUsuario*, *usu*, *ide*, *descripcion*, *localizacion*, *nSeguidores*, *nSeguidos*, *nTweets*, *nFavoritos*, *fotoUsuario*, *comida*, *animales*, *ropa*, *terrorismo*, *sinCalificar*, *privacidad*)

Crea la ventana que contiene la información de un usuario.

Parameters

`nombreUsuario:` contiene el nombre completo del usuario.
(type=str)

`usu:` contiene el nombre de usuario.
(type=str)

`ide:` Contiene el identificador del usuario.
(type=int)

`descripcion:` Contiene la descripción del usuario.
(type=str)

`localizacion:` Contiene la localización del usuario.
(type=str)

`nSeguidores:` Contiene el número de seguidores del usuario.
(type=int)

`nSeguidos:` Contiene el número de seguidos del usuario.
(type=int)

`nTweets:` Contiene el número de tweets del usuario.
(type=int)

`nFavoritos:` Contiene el número de tweets favoritos del usuario.
(type=int)

`fotoUsuario:` Contiene la dirección de la foto de usuario.
(type=str)

`comida:` Contiene el contador de la categoría comida.
(type=int)

`animales:` Contiene el contador de la categoría animales.
(type=int)

`ropa:` Contiene el contador de la categoría ropa.
(type=int)

`terrorismo:` Contiene el contador de la categoría terrorismo.
(type=int)

`sinCalificar:` Contiene el contador de la categoría sinCalificar.
(type=int)

`privacidad:` Contiene si el usuario es privado o no.
(type=boolean)

crearVentanaManejoGrafica(*self*)

Crea la ventana que maneja el Grafo.

crearVentanaCategoria(*self*, *titulo*, *cat*, *contador*, *tweets*)

Crea la ventana que contiene los tweets de una categoría.

Parameters

`titulo:` Variable para cambiar el título de la nueva ventana.

(type=str)

`cat:` Variable que almacena la categoría que mostramos.

(type=str)

`contador:` Número de tweets de la categoría que estudiamos.

(type=int)

`tweets:` Tweets del usuario y de la categoría elegida almacenados en una lista para poder mostrarlos.

(type=list)

crearVentanaError(*self*, *error*)

Crea la ventana que advierte de un error.

Parameters

`error:` Error que se produce para mostrarlo en más detalles.

(type=str)

crearVentanaAlertaUsuario(*self*)

Crea la ventana para mostrar que el nodo Raíz a sido cambiado con éxito.

crearVentanaFinGrafo(*self*)

Crea la ventana para mostrar que el grafo a finalizado con éxito.

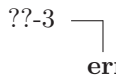
4 Module error

Este módulo contiene la clase **Error**.

Author: Rubén Jiménez Ortega.

Version: 1.0

4.1 Class Error



Crea un QMessageBox (es un diálogo utilizado para mostrar Algún tipo de error).

4.1.1 Methods

___init___(*self*, *texto*)

Constructor con parámetros de la clase **Error**.

Parameters

`texto:` Error que se produce para mostrarlo en más detalles.

(type=str)

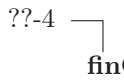
5 Module finGrafo

Este módulo contiene la clase **FinGrafo**.

Author: Rubén Jiménez Ortega.

Version: 1.0

5.1 Class FinGrafo



finGrafo.FinGrafo

Crea un QMessageBox (es un diálogo utilizado para mostrar que el grafo a finalizado).

5.1.1 Methods

___init___(*self*)
Constructor por defecto de la clase **FinGrafo**.

6 Module funcionesTwitter

Este módulo contiene la clase **FuncionesTwitter**.

Author: Rubén Jiménez Ortega.

Version: 1.0

6.1 Variables

Name	Description
archivo_consumer_key	Archivo donde leemos la clave del consumidor. **Value:** `open("../variables/consumer_key.txt", "r")`
consumer_key	Clave del consumidor. **Value:** `archivo_consumer_key.readline() [:-1]`
archivo_consumer_secret	Archivo donde leemos la clave secreta del consumidor. **Value:** `open("../variables/consumer_secret.txt", "r")`
consumer_secret	Clave secreta del consumidor. **Value:** `archivo_consumer_secret.readline() [:-1]`
archivo_access_token	Archivo donde leemos el token de acceso. **Value:** `open("../variables/access_token.txt", "r")`
access_token	Token de acceso. **Value:** `archivo_access_token.readline() [:-1]`
archivo_access_token_secret	Archivo donde leemos el token de acceso secreto. **Value:** `open("../variables/access_token_secret.txt", "r")`
access_token_secret	Token de acceso secreto. **Value:** `archivo_access_token_secret.readline() [:-1]`
auth	Creación de una instancia OAuthHandler. **Value:** `tweepy.OAuthHandler(consumer_key, consumer_secret)`
api	Creación de la interface, usando auth. **Value:** `tweepy.API(auth)`

6.2 Class FuncionesTwitter

Esta clase contiene todos los métodos relacionados con la API de Twitter (tweepy):

- Comprobar si un usuario es privado.
- Escribe los datos de un usuario.
- Escribe los tweets de un usuario.

- Obtiene los tweets separados en las distintas categorías.
- Cuenta y ordena de mayor a menor los tweets de las distintas categorías.
- Lee los distintos diccionarios que han sido creados.

6.2.1 Methods

esPrivado(*self*, *usu*)

Comprueba si el usuario pasado por parámetro es privado o no.

Parameters

`usu:` Usuario al que se somete la comprobación.

(type=str)

Return Value

Devuelve un booleano si el usuario es privado o no.

(type=boolean)

existeUsuario(*self*, *usuario*, *ventanaInicio*)

Comprueba si el usuario pasado por parámetro es privado o no. En caso de que sea privado nos salta una ventana de error avisando de este hecho. En caso de que no sea privado, se da comienzo a la interfaz que maneja el grafo, se obtienen los seguidos y los tweets de ese usuario. Además se coloca este usuario como nodo raíz de nuestro grafo.

Parameters

`usuario:` Usuario al que se somete las operaciones anteriores.

(type=str)

`ventanaInicio:` Ventana en la que hay que insertar el usuario, para poder ocultar dicha ventana.

(type=QWidget)

Raises

`tweepy.TweepError` En el caso de que se inserte un usuario que no exista.

usuarioEstudiado(*self*, *usuario*)

Comprueba si el usuario pasado por parámetro ha sido estudiado con anterioridad, es decir, que haya sido pintado en el Grafo. Si ha sido estudiado se obtienen los datos de este usuario, en caso contrario se genera un error informando que este usuario no ha sido estudiado.

Parameters

`usuario:` Usuario al que se somete las operaciones anteriores.

(type=str)

Raises

`tweepy.TweepError` En el caso de que se inserte un usuario que no exista.

usuarioEstudiadoNodoRaiz(*self*, *usuario*)

Comprueba si el usuario pasado por parámetro es privado, en ese caso, aparece una ventana indicando ese error, en caso contrario, comprobamos si hemos estudiado este usuario, si ha sido estudiado, comprobamos si tenemos sus seguidos, si no los tenemos, los sacamos. Además, cambiamos el nodo Raíz a este usuario y sacamos una ventan alertando este suceso. Si no ha sido estudiado se informa de este error.

Parameters

`usuario:` Usuario al que se somete las operaciones anteriores.

(type=str)

Raises

`tweepy.TweepError` En el caso de que se inserte un usuario que no exista.

seguidos(*self*, *usu*)

Si el usuario no es privado, pasamos a a obtener sus últimos 5000 seguidos (Ya que la API de twitter, nos limita este aspecto, para evitar algunos problemas), estos seguidos serán guardados en un archivo csv para su posterior uso. En caso de ser privado se crea el archivo pero vacio, para tener presente su estudio.

Parameters

`usu:` Usuario al que se somete las operaciones anteriores.

(type=str)

comienzoGrafo(*self*)

Inicia la ventana que hace posible el manejo del Grafo.

obtenerDatos(*self*, *usuario*)

Saca la información de un usuario que ya haya sido estudiado (Nombre de usuario, descripción, localización, número de seguidores, número de seguidos, número de tweets, número de tweets Favoritos, la foto de perfil, el identificador del usuario, si el usuario es privado o no y los contadores de tweets en las diferentes categorías) para mostrarla en la ventana de Información.

Parameters

`usuario:` Usuario al que se somete las operaciones anteriores.

(type=str)

contadorTweetsPorCategorias(*self*, *usu*, *nTweets*)

Esta función se encarga de organizar los tweets por categorías y sus respectivos contadores siempre que el usuario no sea privado, en cuyo caso sus contadores se pondrán a 0 por defecto. Decir también que la limitación que nos da la API de twitter en este caso es que no se podrán obtener más de los 3200 últimos tweets de cada usuario. Para sacar la categoría en la que se clasificará cada tweet, se comprueba los diccionarios que han sido creados con anteridad (estos se pueden aumentar o variar siempre y cuando se crea necesario, editando sus respectivos ficheros) y se compara cada palabra de esos diccionarios con todos los tweets estudiados, si aparece una de las palabras de ese diccionario en un tweet, este será clasificado directamente a esa categoría. Tras esto se procede a escribir los contadores de las categorías en un fichero, que será consultado si se desea ver la información de un usuario. Y los tweets por categorías se escribirá un fichero por cada categoría siempre y cuando el contador sea distinto de 0.

Parameters

`usu:` Usuario al que se somete las operaciones anteriores.

(type=str)

`nTweets:` Número totales de tweets del usuario estudiado.

(type=int)

escribirDatosUsuario(*self*, *usuario*, *contadorComida*, *contadorRopa*, *contadorAnimales*, *contadorTerrorismo*, *contadorSinCalificar*)

Función para guardar los contadores de las categorías en un archivo que sigue la siguiente sintaxis: "usuario_tweets.csv" siendo usuario uno de los parámetros que se pasan a la función. Este archivo a parte de los contadores, se almacena también el identificador del usuario y el nombre del usuario.

Parameters

`usuario:` Nombre del usuario para el nombre y el contenido del archivo nuevo.

(type=str)

`contadorComida:` Variable que contiene el contador de comida.

(type=int)

`contadorRopa:` Variable que contiene el contador de ropa.

(type=int)

`contadorAnimales:` Variable que contiene el contador de animales.

(type=int)

`contadorTerrorismo:` Variable que contiene el contador de terrorismo.

(type=int)

`contadorSinCalificar:` Variable que contiene el contador de de los tweets sin calificar.

(type=int)

escribirTweetsClasificados(*self*, *usu*, *tweetsComida*, *tweetsRopa*, *tweetsAnimales*, *tweetsTerrorismo*, *tweetsSc*)

Función para guardar los tweets de cada categoría en su respectivo archivo. Los tweets vienen en forma de lista y se pasan a los archivos que siguen la siguiente sintaxis: "usu_tweets_*.csv", siendo "usu" el usuario pasado por parámetro y "*" las distintas categorías se pueden dar.

Parameters

`usu:` Nombre del usuario para el nombre de los archivos que se crean.

(type=str)

`tweetsComida:` Variable que contiene la lista de de los tweets de comida.

(type=list)

`tweetsRopa:` Variable que contiene la lista de de los tweets de ropa.

(type=list)

`tweetsAnimales:` Variable que contiene la lista de de los tweets de animales.

(type=list)

`tweetsTerrorismo:` Variable que contiene la lista de de los tweets de terrorismo.

(type=list)

`tweetsSc:` Variable que contiene la lista de de los tweets sin calificar.

(type=list)

ordenarContadores(*self*, *usuario*)

Ordena los contadores de las distintas categorías de un usuario. La prioridad que se sigue en caso de igualdad es la siguiente: terrorismo - animales - comida - ropa, dejando la categoría sin calificar para cuando se de el caso de que todas las demás categorías sean 0 o bien cuando el usuario sea privado y no podamos acceder a sus tweets.

Parameters

`usuario:` Usuario al que queremos conocer su contador mayor.

(type=str)

Return Value

Devuelve **valor** en función de que categoría que tenga mayor contador. Los valores que se pueden dar son los siguientes: Si *valor = 0* –> categoría sin calificar, si *valor = 1* –> categoría terrorismo, si *valor = 2* –> categoría animales, si *valor = 3* –> categoría comida, si *valor = 4* –> categoría Ropa.

(type=int)

leerContadores(*self*, *usuario*)

Esta función se encarga de leer los contadores que se encuentran almacenados en los ficheros "usuario_tweets.csv", siendo *usuario* el parámetro que pasamos a esta función.

Parameters

`usuario:` Usuario al que queremos conocer sus contadores.

(type=str)

Return Value

Devuelve los valores de los contadores de **comida**, **ropa**, **animales**, **terrorismo** y **sinCalificar**.

(type=int)

obtenerTweetsAnimales(*self*, *usu*)

Esta función se encarga de obtener los tweets relacionados con los animales para el usuario **usu**. Para ello, se llamamos a la función *leerTweetsAnimales(usu)*, y una vez leidos, creamos la ventana con estos tweets.

Parameters

`usu:` Usuario del que queremos conocer sus tweets de animales.

(type=str)

leerTweetsAnimales(*self*, *usu*)

Esta función se encarga de leer los tweets de la categoría animales para el usuario **usu**. Una vez que se accede al archivo, estos tweets, serán almacenados en una lista para después poder mostrarlos. En caso de que el archivo que buscamos no exista, la lista quedará vacia, o lo que es lo mismo, que no hay tweets en esta categoría.

Parameters

`usu:` Usuario al que queremos leer sus tweets de la categoría animales.

(type=str)

Return Value

Devuelve una lista **animales**.

(type=list)

obtenerTweetsComida(*self*, *usu*)

Esta función se encarga de obtener los tweets relacionados con la comida para el usuario **usu**. Para ello, se llamamos a la función *leerTweetsComida(usu)*, y una vez leidos, creamos la ventana con estos tweets.

Parameters

`usu:` Usuario del que queremos conocer sus tweets de comida.

(type=str)

leerTweetsComida(*self*, *usu*)

Esta función se encarga de leer los tweets de la categoría comida para el usuario **usu**. Una vez que se accede al archivo, estos tweets, serán almacenados en una lista para después poder mostrarlos. En caso de que el archivo que buscamos no exista, la lista quedará vacia, o lo que es lo mismo, que no hay tweets en esta categoría.

Parameters

`usu:` Usuario al que queremos leer sus tweets de la categoría comida.

(type=str)

Return Value

Devuelve una lista **comida**.

(type=list)

obtenerTweetsRopa(*self*, *usu*)

Esta función se encarga de obtener los tweets relacionados con la ropa para el usuario **usu**. Para ello, se llamamos a la función *leerTweetsRopa(usu)*, y una vez leidos, creamos la ventana con estos tweets.

Parameters

`usu:` Usuario del que queremos conocer sus tweets de ropa.

(type=str)

leerTweetsRopa(*self*, *usu*)

Esta función se encarga de leer los tweets de la categoría ropa para el usuario **usu**. Una vez que se accede al archivo, estos tweets, serán almacenados en una lista para después poder mostrarlos. En caso de que el archivo que buscamos no exista, la lista quedará vacia, o lo que es lo mismo, que no hay tweets en esta categoría.

Parameters

`usu:` Usuario al que queremos leer sus tweets de la categoría ropa.

(type=str)

Return Value

Devuelve una lista **ropa**.

(type=list)

obtenerTweetsTerrorismo(*self*, *usu*)

Esta función se encarga de obtener los tweets relacionados con el terrorismo para el usuario **usu**. Para ello, se llamamos a la función *leerTweetsTerrorismo(usu)*, y una vez leidos, creamos la ventana con estos tweets.

Parameters

`usu:` Usuario del que queremos conocer sus tweets de terrorismo.

(type=str)

leerTweetsTerrorismo(*self*, *usu*)

Esta función se encarga de leer los tweets de la categoría terrorismo para el usuario **usu**. Una vez que se accede al archivo, estos tweets, serán almacenados en una lista para después poder mostrarlos. En caso de que el archivo que buscamos no exista, la lista quedará vacia, o lo que es lo mismo, que no hay tweets en esta categoría.

Parameters

`usu:` Usuario al que queremos leer sus tweets de la categoría terrorismo.

(type=str)

Return Value

Devuelve una lista **terrorismo**.

(type=list)

obtenerTweetsSc(*self*, *usu*)

Esta función se encarga de obtener los tweets relacionados sin calificar para el usuario **usu**. Para ello, se llamamos a la función *leerTweetsSc(usu)*, y una vez leidos, creamos la ventana con estos tweets.

Parameters

`usu:` Usuario del que queremos conocer sus tweets sin calificar.

(type=str)

leerTweetsSc(*self*, *usu*)

Esta función se encarga de leer los tweets de la categoría sin calificar para el usuario **usu**. Una vez que se accede al archivo, estos tweets, serán almacenados en una lista para después poder mostrarlos. En caso de que el archivo que buscamos no exista, la lista quedará vacia, o lo que es lo mismo, que no hay tweets en esta categoría.

Parameters

`usu:` Usuario al que queremos leer sus tweets de la categoría sin calificar.

(type=str)

Return Value

Devuelve una lista **sc**.

(type=list)

obtenerDiccionarioAnimales(*self*)

Esta función se encarga de leer el diccionario de animales y almacenar sus términos en una lista.

Return Value

Devuelve **listaAnimales**.

(type=list)

obtenerDiccionarioRopa(*self*)

Esta función se encarga de leer el diccionario de ropa y almacenar sus términos en una lista.

Return Value

Devuelve **listaRopa**.

(type=list)

obtenerDiccionarioComida(*self*)

Esta función se encarga de leer el diccionario de comida y almacenar sus términos en una lista.

Return Value

Devuelve **listaComida**.

(type=list)

obtenerDiccionarioTerrorismo(*self*)

Esta función se encarga de leer el diccionario de terrorismo y almacenar sus términos en una lista.

Return Value

Devuelve **listaTerrorismo**.

(type=list)

obtenerTweetsUsuario(*self*, *usuario*)

Esta función se encarga de leer todos los tweets de un usuario siempre que sea permitido por la API.

Parameters

`usuario:` Usuario del que queremos obtener sus tweets.

(type=str)

Return Value

Devuelve **salida**, con los tweets de un usuario dado.

(type=list)

7 Module grafo

Este módulo contiene la clase **Grafo**.

Author: Rubén Jiménez Ortega.

Version: 1.0

7.1 Variables

Name	Description
ventanaManejoGrafo	Esta varible almacenará la ventana que maneja el Grafo, para así poder cambiar el estado de los botones. **Value:** `ManejoGrafo()`
terminado	Esta varible es falsa normalmente, salvo que se termine el estudio de la lista de seguidos de un usuario. **Value:** `False`
inicio	Esta varible siempre vale True a no ser que se pause, pare o termine el grafo. **Value:** `True`
G	Inicialización del Grafo. **Value:** `nx.Graph()`
numeroPausa	Para saber por que posición vamos en caso de que se pause el grafo. **Value:** `0`
nodoRaiz	Valor del nodo raíz. O bien se pasa al inicio del programa o bien cuando cambiamos de nodo. **Value:** `""`
segundoNivel	Cuando estamos estudiando el segundo nivel, el usuario. **Value:** `""`
row_count	Número de columnas que tiene un usuario en sus seguidos, o lo que es lo mismo, el número de seguidos. **Value:** `0`
row_count_nodo_raiz	Número de columnas que tiene el usuario raíz en sus seguidos, o lo que es lo mismo, el número de seguidos. **Value:** `0`
contador	Para ver por donde vamos. **Value:** `0`
numeroAlgunasCategorias	Esta varible almacena el numero de usuarios que hay cuando no se seleccionan todas la categorias. **Value:** `0`
listaNodos	Se van añadiendo los nodos cuando no se han seleccionado todas las categorías. **Value:** `[]`
fin	Indica cuando hemos terminado el grafo. **Value:** `False`

7.2 Class Grafo

Esta clase contiene todos los métodos relacionados con el Grafo.

7.2.1 Methods

setInicio(*self*, *inicioNuevo*)

Modifica el valor de la variable inicio.

Parameters

`inicioNuevo:` Nueva valor que adquiere *inicio*.

(type=boolean)

setNodoRaiz(*self*, *nuevoNodo*)

Modifica el valor del nodo Raíz.

Parameters

`nuevoNodo:` Nueva valor que adquiere *nodoRaiz*.

(type=str)

segundoNivel(*self*, *nuevoSegundoNivel*)

Modifica el valor del segundoNivel.

Parameters

`nuevoSegundoNivel:` Nueva valor que adquiere *segundoNivel*.

(type=str)

setNumeroAlgunasCategorias(*self*, *numero*)

Modifica el valor de numeroAlgunasCategorias.

Parameters

`numero:` Nueva valor que adquiere *numeroAlgunasCategorias*.

(type=int)

setNumero(*self*, *numeroNuevo*)

Modifica el valor de numero.

Parameters

`numeroNuevo:` Nueva valor que adquiere *numero*.

(type=int)

setContador(*self*, *contadorNuevo*)

Modifica el valor de contador.

Parameters

`contadorNuevo:` Nueva valor que adquiere *contador*.

(type=int)

leerDocumentoTodasCategorias(*self*, *G*, *numero*, *row_count*, *usu*)

Lee la lista de seguidos del usuario que se pasa por parámetro, tras esto, en cada iteracción del bucle va añadiendo una relación nueva que se añade al grafo. Tras acabar la iteracción se sale de la función para hacer que el grafo vaya creciendo progresivamente. Cuando termina la lista de un usuario pone terminado a True y pasa al segundo nivel.

Parameters

`G:` Grafo al que añadiremos los nodos.

(type=nx.Graph())

`numero:` Para saber por donde vamos.

(type=int)

`row_count:` Número de columnas en el archivo del usuario en cuestión.

(type=int)

`usu:` Usuario del cual leemos sus seguidos.

(type=str)

sacarSeguidosSegundoNivel(*self*, *usuario*)

Leemmos los seguidos del usuario pasado por parámetro. A partir de estos, se irán sacando las listas de los usuarios a segundo nivel. Una vez terminado, rompemos la interacción.

Parameters

`usuario:` Usuario a partir del cual leemos sus seguidos, y de estos sacamos sus seguidos.

(type=str)

leerDocumentoAlgunasCategorias(*self*, *G*, *row_count*, *usu*, *checkAnimales*, *checkRopa*, *checkTerrorismo*, *checkComida*, *checkSc*)

Lee la lista de seguidos del usuario que se pasa por parámetro, tras esto, se chequean que botones están activos para contar como una iteracción un usuario que cumpla los requisitos, si dicho usuario no los cumple pasa al siguiente, pero a diferencia de la función anterior, esta no se contaría como interacción. Cuando termina la lista de un usuario pone terminado a True y pasa al segundo nivel.

Parameters

`G:` Grafo al que añadiremos los nodos.
(type=nx.Graph())

`row_count:` Número de columnas en el archivo del usuario en cuestión.
(type=int)

`usu:` Usuario del cual leemos sus seguidos.
(type=str)

`checkAnimales:` Para saber si esta activado o no el checkBox de animales.
(type=boolean)

`checkRopa:` Para saber si esta activado o no el checkBox de ropa.
(type=boolean)

`checkTerrorismo:` Para saber si esta activado o no el checkBox de terrorismo.
(type=boolean)

`checkComida:` Para saber si esta activado o no el checkBox de comida.
(type=boolean)

`checkSc:` Para saber si esta activado o no el checkBox de sin calificar.
(type=boolean)

sacarSeguidosSegundoNivelAlgunasCategorias(*self*, *usuario*, *G*)

Leemmos la lista de nodos que hay en G. Por cada iteracción iremos borrando de la lista el nodo consultado y sacando sus seguidos para posterior estudio.

Parameters

`usuario:` Usuario a partir del cual leemos sus seguidos, y de estos sacamos sus seguidos.
(type=str)

`G:` Grafo al que consultamos sus nodos.
(type=nx.Graph())

colorearNodos(*self*, *G*, *color_map*)

Se colorean los nodos en función del valor de los contoadores de cada usuario. Los colores serían:

- **Gris**: Sin calificar.
- **Rojo**: Terrorismo.
- **Verde**: Animales.
- **Azul**: Comida.
- **Amarillo**: Ropa.

Parameters

`G:` Grafo al que consultamos sus nodos.
(type=nx.Graph())

`color_map:` Lista para colorear los nodos.
(type=list)

borrarGrafo(*self*)

Borra el Grafo.

definirRowCount(*self*, *usu*)

Define el número de columnas que hay en el archivo de seguidos del usuario **usu**.

Parameters

`usu:` Usuario al que queremos sacar las columnas de su archivo de seguidos.
(type=str)

Return Value

row_count.
(type=int)

terminar(*self*)

Bloquea los botones al terminar el Grafo por completo.

errorSinCategoria(*self*)

Muestra un error en caso de que no se seleccione ninguna categoría.

dibujar(*self*, *checkAnimales*, *checkRopa*, *checkTerrorismo*, *checkComida*, *checkSc*, *manejoGrafo*)

Esta función se divide en tres partes. La primera, si todos los checkBox están activados, en tal caso se llaman a las funciones *leerDocumentoTodasCategorias* y *sacarSeguidosSegundoNivel* para el correcto funcionamiento del grafo. La segunda, no todos los checkBox están activados, en tal caso se llaman a las funciones *leerDocumentoAlgunasCategorias* y *sacarSeguidosSegundoNivelAlgunasCategorias* para el correcto funcionamiento del grafo. Y por último, si no hay ningún checkBox activado, en tal caso se procede al error.

Parameters

`checkAnimales:` Para saber si esta activado o no el checkBox de animales.
(type=boolean)

`checkRopa:` Para saber si esta activado o no el checkBox de ropa.
(type=boolean)

`checkTerrorismo:` Para saber si esta activado o no el checkBox de terrorismo.
(type=boolean)

`checkComida:` Para saber si esta activado o no el checkBox de comida.
(type=boolean)

`checkSc:` Para saber si esta activado o no el checkBox de sin calificar.
(type=boolean)

`manejoGrafo:` Ventana de manejoGrafo para manejar el comportamiento de sus componentes.
(type=object)

8 Module informacion

Este módulo contiene la clase **Informacion**.

Author: Rubén Jiménez Ortega.

Version: 1.0

8.1 Class Informacion

object
└ **informacion.Informacion**

Crea un object para poder manejar la información del usuario.

8.1.1 Methods

setupUi(*self*, *Form*)

Actualiza la ventana que se le pasa por parámetro añadiendole los componentes.

Parameters

`Form:` Ventana.

retranslateUi(*self*, *Form*, *nombre*, *alias*, *ide*, *descripcion*, *ubicacion*, *seguidores*, *siguiendo*, *tweets*, *fav*, *foto*, *comida*, *animales*, *ropa*, *terrorismo*, *Sc*, *privacidad*)

Actualiza la ventana que se le pasa por parámetro añadiendole los verdaderos valores de los componentes.

Parameters

`Form:`	Ventana
`nombre:`	contiene el nombre completo del usuario. *(type=str)*
`alias:`	contiene el nombre de usuario. *(type=str)*
`ide:`	Contiene el identificador del usuario. *(type=int)*
`descripcion:`	Contiene la descripción del usuario. *(type=str)*
`ubicacion:`	Contiene la localización del usuario. *(type=str)*
`seguidores:`	Contiene el número de seguidores del usuario. *(type=int)*
`siguiendo:`	Contiene el número de seguidos del usuario. *(type=int)*
`tweets:`	Contiene el número de tweets del usuario. *(type=int)*
`fav:`	Contiene el número de tweets favoritos del usuario. *(type=int)*
`foto:`	Contiene la dirección de la foto de usuario. *(type=str)*
`comida:`	Contiene el contador de la categoría comida. *(type=int)*
`animales:`	Contiene el contador de la categoría animales. *(type=int)*
`ropa:`	Contiene el contador de la categoría ropa. *(type=int)*
`terrorismo:`	Contiene el contador de la categoría terrorismo. *(type=int)*
`Sc:`	Contiene el contador de la categoría sinCalificar. *(type=int)*
`privacidad:`	Contiene si el usuario es privado o no. *(type=boolean)*

pulsarComida(*self*, *event*)

Abre la ventana categoría comida.

Parameters

`event:` evento.

pulsarAnimales(*self*, *event*)

Abre la ventana categoría animales.

Parameters

`event:` evento.

pulsarRopa(*self*, *event*)

Abre la ventana categoría ropa.

Parameters

`event:` evento.

pulsarTerrorismo(*self*, *event*)

Abre la ventana categoría terrorismo.

Parameters

`event:` evento.

pulsarSc(*self*, *event*)

Abre la ventana categoría sin calificar.

Parameters

`event:` evento.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___init___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

8.1.2 Properties

Name	Description
Inherited from object	
___class___	

9 Module inicio

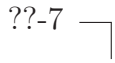
Este módulo contiene la clase **VentanaInicio**.

Author: Rubén Jiménez Ortega.

Version: 1.0

Copyright: Copyright (C) 2017 by Rubén Jiménez Ortega.

9.1 Class VentanaInicio



Crea un cuadro de texto para insertar el nombre de usuario para lanzar la aplicación, en caso de que este todo correcto.

9.1.1 Methods

___init___(*self*, **args*)

Constructor por defecto de la clase **VentanaInicio**.

comprobar(*self*, *cuadroTexto*)

Función para comprobar si un usuario exite o no.

Parameters

`cuadroTexto`: Cuadro donde se inserta el usuario.
(type=QLineEdit())

borrarContenidoCuadro(*self*, *cuadroTexto*)

Borra el contenido de un cuadro de texto (QLineEdit)

Parameters

`cuadroTexto`: Cuadro que se borrará.
(type=QLineEdit())

center(*self*)

Función para centrar cualquier ventana en una pantalla, independientemente del tamaño de la misma.

10 Module manejoGrafo

Este módulo contiene la clase **ManejoGrafo**.

Author: Rubén Jiménez Ortega.

Version: 1.0

10.1 Class ManejoGrafo

object
└ **manejoGrafo.ManejoGrafo**

Crea un object para poder manejar el Grafo.

10.1.1 Methods

setupUi(*self*, *Form*)

Actualiza la ventana que se le pasa por parámetro añadiendole los componentes.

Parameters
`Form`: Ventana.

retranslateUi(*self*, *Form*)

Actualiza la ventana que se le pasa por parámetro añadiendole los verdaderos valores de los componentes.

Parameters
`Form`: Ventana.

bloquearBotonesTerminar(*self*)

Bloquea los botones necesarios tras la finalización del grafo.

errorNoCategorias(*self*)

Muestra el error de que no hay categorías seleccionadas y además desbloquea y bloquea lo que se estima necesario.

comprobar(*self*, *cuadroTexto*)

Esta función llama a *usuarioEstudiado* de la clase {funcionesTwitter} para ver si ha sido estudiado el usuario con anterioridad.

Parameters

`cuadroTexto`: Cuadro donde se inserta el usuario.

(type=QLineEdit())

cambioNodo(*self*, *cuadroTexto*)

Esta función llama a *usuarioEstudiadoNodoRaiz* de la clase {funcionesTwitter} para ver si ha sido estudiado el usuario con anterioridad y poder cambiarlo a nodo raíz o no.

Parameters

`cuadroTexto`: Cuadro donde se inserta el usuario.

(type=QLineEdit())

salir(*self*)

Cierra el programa y borra todos los archivos que se han creado.

inicio(*self*)

Da comienzo al grafo o lo continua si ha sido pausado, bloqueando los componentes necesarios para su correcto uso.

pausa(*self*)

Pausa el grafo y bloquea los componentes necesarios para su correcto uso.

parar(*self*)

Detiene y borra el grafo y bloquea los componentes necesarios para su correcto uso.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___init___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.1.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
___class___	

Index