PoliCal

Release 1.1.3

andr3slelouch

Feb 17, 2021

CONTENTS:

CHAPTER

ONE

POLICAL MAIN

polical.__main__.**define_args**()
This function defines the arguments for cli

Returns: args(Namespace): The namespace with the defined arguments

CHAPTER

TWO

POLICAL GET_TRELLO_MOODLEEPN_KEYS

class polical.Get_Trello_MoodleEPN_Keys.**DevNullRedirect**
Temporarily eat stdout/stderr to allow no output. This is used to suppress browser messages in webbrowser.open

polical.Get_Trello_MoodleEPN_Keys.**check_file_existence**(*output_file: str*) → bool
This functions checks if the file already exists on filesystem

Args: output_file (str): File location

Returns: bool. The return code:

```
False -- If the file not exists or is blank
True -- If already exists
```

polical.Get_Trello_MoodleEPN_Keys.**check_user_on_file**(*output_file: str*, *username: str*)
This functions checks if an user already exists on yaml file.

Args: output_file (str): File location username (str): The username that would be checked.

Returns: Stop onboard function if the username already exists and the user answers N to overwrite it.

polical.Get_Trello_MoodleEPN_Keys.**get_working_board_id**(*api_key: str*, *api_secret: str*, *oauth_token: str*, *oauth_token_secret: str*) -> (<*class 'str'*>, <*class 'str'*>)
This function is for getting the TareasPoli Board ID that is created in the onboard process

Args: api_key (str): The api key to acceso Trello api_secret (str): The api secret to acceso Trello oauth_token (str): The oauth token to acceso Trello oauth_token_secret (str): The oauth token secret to acceso Trello

Returns: board_id (str): This is the board id of TareasPoli board owner_id (str): This is the owner id of Trello user

polical.Get_Trello_MoodleEPN_Keys.**onboard**(*no_open: bool*, *output_path='polical.yaml'*)
Obtain Trello API credentials and put them into your config file. This is invoked automatically the first time you attempt to do an operation which requires authentication. The configuration file is put in an appropriate place for your operating system.

CHAPTER

THREE

POLICAL CONFIGURATION

polical.configuration.**add_subject_to_trello_list**(*subjects_board*, *subject_name: str*, *subject_code: str*, *username: str*)

This function adds a subject as list to trello board.

Args: subjects_board (Trello.Board): Tareas Poli's Board object from Trello library subject_name (str): Subject name to create new list. subject_code (str): Subject code to add it to name list. username (str): The username for the owner of the current subject.

polical.configuration.**check_for_url**(*url: str*) → bool

This function is for checking moodle calendar url

Args: url (str): Moodle Calendar Address to be checked

Returns: bool. The return code:

```
False -- If the url does not start with https and ends with recentupcoming
True -- If the url is correct
```

polical.configuration.**create_subject**(*subject_code: str*, *task_title: str*, *user_dict: dict*, *username: str*, *trello_account=True*) → bool

This function creates a subject in Trello if trello_account is True, and adds it to local database associating to a user.

Args: subject_code (str): Subject Code to check with local database and trello. task_title (str): Subject title for showing to the user if subject is not founded in local database. user_dict (dict): User dictionary with keys to connect to trello. username (str): The username for the owner of the current subject. trello_account (bool): Flag to indicate if the subject would be created in Trello.

Returns: bool. The return code:

```
False -- If the Trello account flag is deactivated and the subject is not␣
↪founded in local database
True -- If the process to create the subject was successful
```

polical.configuration.**generate_db_selector_file**(*config_file_path: str*)

This function generates a db selector file template with the required parameters

Args: config_file_path (str): Where the file should be stored

polical.configuration.**get_bot_token**(*config_file_path: str*) → str

Returns the bot token stored in config_file_path

Args: config_file_path (str): Where the file is stored

Returns: str: Token to access to Telegram Bot

polical.configuration.**get_file_location**(*filename: str*) → str

This function gets the full path location of a file

Args: filename (str): Filename that needs the full path location

Returns: full_path_file_location (str): Full path location of the file

polical.configuration.**get_mysql_credentials**(*config_file_path: str*) → dict
This function returns the credentials to access to mysql database stored in config.yaml config file

Args: config_file_path (str): Where is the file stored

Returns: dict: Dictionary that contains username and password for the MySQL database

polical.configuration.**get_preferred_dbms**(*config_file_path: str*) → str
Returns the dbms to be used stored in config_file_path

Args: config_file_path (str): Where the file is stored

Returns: str: The database that is used

polical.configuration.**get_subject_name_from_ics_event_category**(*full_subject_name: str*) → str
This function gets subject name from the categories in ics event

Args: full_subject_name (str): Full subject name with format XXX_YYY_ZZZ

Return: subject_name (str): Subject name

polical.configuration.**get_working_directory**() → str
This functions gets the working directory path.

Returns: working_directory (str): The directory where database and yaml are located.

polical.configuration.**load_config_file**(*config_file_path: str*) → dict
This function is for loading yaml config file

Args: config_file_path (str): File path for the config file to be loaded

Returns: file_config (dict): Dictionary with config keys.

Raises: IOError

polical.configuration.**prepare_mysql_query**(*query: str*) → str
This function changes ? characters to %s for being used by MySQL Connection

Args: query (str): Database query to execute

Returns: str: SQLite3 query parsed to MySQL query

polical.configuration.**set_bot_token**(*config_file_path: str*, *token: str*)
Saves the telegram token for being used for the bot telegram

Args: config_file_path (str): Where the file is stored token (str): A valid token to access to Telegram Bot

polical.configuration.**set_preffered_dbms**(*preffered_dbms: str*, *config_file_path: str = '/home/docs/PoliCal/config.yaml'*)
Saves the database that polical is using

Args:

preffered_dbms (str):

- mysql: To use MySQL database
- default: To use SQLite3 database

config_file_path (str, optional): Where the file is stored . Defaults to "config.yaml".

CHAPTER

FOUR

POLICAL SENDTASKTOTRELLO

polical.SendTaskToTrello.**send_task_to_trello**(*username: str*, *user_dict: dict*)

This function sends tasks from database that are stored as not sended to trello.

Args: username (str): The username for the current task. user_dict (dict): User dictionary with keys to acces to trello.

CHAPTER

FIVE

POLICAL SIMPLEICSTOCSV

polical.SimpleIcsToCSV.**add_event**(*header: list*, *filename: str*)
This function adds a event as a new line in csv file.

Args: header (list): Header for the csv. filename (str): The filename where file would be written.

Returns: None. If not has headers

polical.SimpleIcsToCSV.**convert_ics_to_csv**(*url: str*)
This function downloads the moodle calendar and addEvents to a CSV file.

Args: url (str): URL to download moodle calendar

polical.SimpleIcsToCSV.**find_header**(*filename: str*) → list
This function looks for all the file to get the most longest header.

Args: icsCal (str): The ics file location.

Returns: (list): List containing the largest header list.

CHAPTER

SIX

POLICAL TAREASCSVTOBD

polical.TareasCSVToBD.**get_subject_name_from_csv**(*full_subject_name*)
This function gets subject name from csv

Args: full_subject_name (str): Full subject name with format XXX_YYY_ZZZ

Return: subject_name (str): Subject name

polical.TareasCSVToBD.**load_csv_tasks_to_db**(*username: str*, *user_dict: dict*)
This function loads csv tasks to the database

Args: username (str): The username for the current task. user_dict (dict): User dictionary with keys to acces to trello.

Raises: FileNotFoundError

CHAPTER

SEVEN

POLICAL MATERIASLOADERTODB

polical.MateriasLoaderToDB.**load_subjects_to_db**()
: This function loads any subject located on materias.csv to the database.

polical.MateriasLoaderToDB.**update_subjects_to_db**()
: This function loads any subject located on materias.csv to the database.

CHAPTER

EIGHT

POLICAL CONNECTSQLITE

polical.connectSQLite.**add_task_tid**(*task_uid: str*, *task_tid: str*, *username: str*)
This function adds the task trello ID into the database

Args: task_uid (str): Task UID from ICS file. task_tid (str): New Task Trello ID from trello. username (str): Username from the user that owns the task

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**check_no_subject_id**(*subject_code: str*, *username: str*) → bool
This function checks if the subject code is registered and has an ID in the database.

Args: subject_code (str): Subject code from the database to check if it has ID or not.

Returns: (bool): Returns 'False' if does not has the ID and 'True' if it has it.

polical.connectSQLite.**check_task_existence**(*task*) → bool
This function checks if a task exists in the database

Args: task (Tarea): Tasks that would be added to the database.

Returns: bool: If exits True if not False

polical.connectSQLite.**check_user_existence**(*username: str*) → bool
This function checks if the username has an ID in the database.

Args: username (str): username from the database to check if it has ID or not.

Returns: False: If does not has the ID True: If it has it.

polical.connectSQLite.**check_user_subject_existence**(*subject_id: str*, *username: str*) → bool
This function checks if a subject exists in the database and it is associated to a user

Args: subject_id (str): Subject id to be checked username(str): User owner of the task with subjects associated to.

Returns: (bool): Returns True if exists and False if not

polical.connectSQLite.**check_user_task_existence**(*task*, *username: str*)
This function checks if a task exists in the database

Args: task (TareaClass.Tarea): Tasks that would be added to the database. username(str): User owner of the task.

polical.connectSQLite.**exec**(*command: str*)
This function executes a coomand in the database.

Args: command (str): Query that needs to be executed on the database.

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**get_all_users_with_URL**() → list
This function returns all the users that has calendar url registered

Returns: list: Contains a list of users containing username and calendar url

polical.connectSQLite.**get_cur**()
This function returns the database cursor

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**get_db**()
This function returns the database connection. Selects between sqlite3 or mysql

Returns: db (Connection): Database connection that access to tasks and subjects.

polical.connectSQLite.**get_sended_tasks_for_bot**(*username: str*, *message_date: datetime.datetime*) → list
This function gets all sended tasks from the user and with due after message_date.

Args: username (str): Username from the user that owns the task message_date (datetime): Tasks should be after this date

Returns: tasks (list): Database cursor that access to tasks and subjects.

polical.connectSQLite.**get_subject_from_id**(*subject_id*) → *polical.MateriaClass.Materia*
This function gets a subject object from the database

Args: subject_code (str): Subject code for get the subject name.

Returns: subject (MateriaClass.Materia): The subject name from the subject code.

polical.connectSQLite.**get_subject_id**(*subject_code: str*) → str
This function gets the subject ID from the database

Args: subject_code (str): Subject code from the subject to get the ID.

Returns: subject_id (str): Subject ID from the subject.

polical.connectSQLite.**get_subject_name**(*subject_code: str*) → str
This function gets the subject Name from the database

Args: subject_code (str): Subject code for get the subject name.

Returns: subject_name (str): The subject name from the subject code.

polical.connectSQLite.**get_task_id**(*task_uid: str*) → str
This function gets the task ID from the database

Args: task_uid (str): Task UID from the task to get the ID.

Returns: task_id (str): Task ID from the task.

polical.connectSQLite.**get_tasks_for_bot**(*username: str*, *message_date: datetime.datetime*) → list
This function gets all unsended tasks from the user.

Args: username (str): Username from the user that owns the task message_date (datetime): Tasks should be after this date

Returns: tasks (list): Database cursor that access to tasks and subjects.

polical.connectSQLite.**get_unsended_tasks**(*username: str*) → list
This function gets all unsended tasks from the user.

Args: username (str): Username from the user that owns the task

Returns: tasks (list): Database cursor that access to tasks and subjects.

polical.connectSQLite.**get_user_calendar_url**(*username: str*) → str
This function gets the subject ID from the database

Args: subject_code (str): Subject code from the subject to get the ID.

Returns: subject_id (str): Subject ID from the subject.

polical.connectSQLite.**get_user_id**(*username: str*) → str
This function gets the User ID from the database

Args: username (str): Username to get his ID.

Returns: user_id (str): The user id from the database.

polical.connectSQLite.**get_user_subject_name**(*subject_id: str*, *username: str*) → str
Returns the subject name

Args: subject_id (str): ID associated to the subject username (str): Username that has tasks from this subject

Returns: subject_name (str): Subject name

polical.connectSQLite.**save_subject**(*subject:* polical.MateriaClass.Materia)
This function saves a subject into the database

Args: subject (MateriaClass.Materia): Subject that would be added to the database.

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**save_subject_id**(*subject:* polical.MateriaClass.Materia)
DEPRECATED This function saves the trello list ID into the database

Args: subject (MateriaClass.Materia): Subject that owns the ID that would be added to the database.

polical.connectSQLite.**save_task**(*task*)
This function saves a task into the database

Args: task (TareaClass.Tarea): Tasks that would be added to the database.

polical.connectSQLite.**save_user**(*username: str*)
This function saves a user into the database

Args: username (str): User to be added into the database

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**save_user_calendar_url**(*calendar_url: str*, *username: str*)
This function saves a calendar_url to a user

Args: calendar_url (str): Calendar url to ics username(str): User owner of the calendar.

polical.connectSQLite.**save_user_subject**(*subject:* polical.MateriaClass.Materia, *username: str*)
This function saves a subject and associates to a user into the database

Args: subject (MateriaClass.Materia): Subject that would be added to the database. username(str): User owner of the task.

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**save_user_subject_name**(*subject:* polical.MateriaClass.Materia, *username: str*)
This function saves a name for subject into MateriasUsuarios table

Args: subject (MateriaClass.Materia): Subject that would be added to the database. username(str): User owner of the task.

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

polical.connectSQLite.**save_user_task**(*task*, *username: str*)
This function saves a task from a user into the database

Args: task (TareaClass.Tarea): Tasks that would be added to the database. username(str): User owner of the task.

polical.connectSQLite.**update_subject**(*subject:* polical.MateriaClass.Materia)
This function saves a subject into the database

Args: subject (MateriaClass.Materia): Subject that would be added to the database.

Returns: cur (Cursor): Database cursor that access to tasks and subjects.

CHAPTER

NINE

POLICAL POLICALBOT

polical.bot.policalbot.**callback_reminder_message**(*context: telegram.ext.callbackcontext.CallbackContext*) → None

This callback handler registers a new job for remind the user about a task 30 minutes before due the task

polical.bot.policalbot.**error_handler**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

Log the error and send a telegram message to notify the developer.

polical.bot.policalbot.**get_jobs**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

This function is made to kno how much jobs are currently executing

polical.bot.policalbot.**get_moodle_epn_url**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

This command handler is made for receiving the moodle url from the user and saves it to the database

polical.bot.policalbot.**get_new_tasks**(*context: telegram.ext.callbackcontext.CallbackContext*) → None

This functions repeats in specific times, looks for new tasks for every user that has a valid url registered in the database, also defines new jobs for reminder incoming tasks for the users

polical.bot.policalbot.**get_tasks**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

This command handler is made for updating and sending the tasks to the users

polical.bot.policalbot.**save_subject_command**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

This command handler saves a subject if is not registerd in the database

polical.bot.policalbot.**send_new_tasks**(*context: telegram.ext.callbackcontext.CallbackContext*, *username: str*, *tasks: list*) → None

This function sends the tasks as messages to the users also registers josb for new task and remidner to the user

Args: context (CallbackContext): Context sended by the main command handler username (str): username to send the message tasks (list): List of tasks to be sended to the user

polical.bot.policalbot.**start**(*update: telegram.update.Update*, *context: telegram.ext.callbackcontext.CallbackContext*) → None

This command handler starts the bot interactions it sends a message to the user with the instructions to use the bot

CHAPTER

TEN

POLICAL MATERIACLASS

class polical.MateriaClass.**Materia**(*name: str*, *codigo: str*, *id: str = ''*)

Class that stores subject attributes like code and id

print()

Prints a summary of the main attributes of the class

CHAPTER

ELEVEN

POLICAL TAREACLASS

class polical.TareaClass.**Tarea**(*id: str*, *title: str*, *description: str*, *due_date: datetime.datetime*, *subject_id: str*)

Main class of PoliCal it manages the task its owner and its subject associated

define_subject(*subject:* polical.MateriaClass.Materia)

Defines a subject object for the task

Args: subject (MateriaClass.Materia): [description]

define_tid(*tid: str*)

Defines Trello o Telegram ID associated to this task

Args: tid (str): Trello o Telegram ID associated to this task

define_username(*username: str*)

Defines task owner username

Args: username (str): The username of the user that owns the task

print()

Prints a summary ofthe main elements of the task

summary()

Generate a Summary of the main elements of the task

Returns: str: Summary of the task generaly for being sended as Telegram Message

CHAPTER

TWELVE

POLICAL TASKS_PROCESSOR

polical.tasks_processor.**save_tasks_to_db**(*url: str*, *username: str*, *user_dict: dict*, *trello_account=True*)

Save incoming tasks to the database

Args: url (str): ICS url for look for new tasks username (str): User owner of the tasks user_dict (dict): Dictionary that has user configurations trello_account (bool, optional): If tasks will be sended to trello. Defaults to True.

polical.tasks_processor.**send_tasks_to_trello**(*username: str*, *user_dict: dict*)

This function sends tasks from database that are stored as not sended to trello.

Args: username (str): The username for the owner of the tasks. user_dict (dict): User dictionary with keys to acces to trello.

CHAPTER

THIRTEEN

POLICAL TODO_GENERATOR

polical.todo_generator.**get_cards_urls**() → list
This function gets a all uncompleted cards from a board.

Returns: uncompleted_cards (List): List of strings from all uncompleted cards

polical.todo_generator.**get_done_tasks**() → list
This function gets a all tasks that exits on todo.txt.

Returns: tasks_list (List): List of strings from all tasks that exits on todo.txt

polical.todo_generator.**get_todo_tasks**() → list
This function gets a all tasks that exits on todo.txt.

Returns: tasks_list (List): List of strings from all tasks that exits on todo.txt

polical.todo_generator.**get_trello_client**(*user_dict: dict*) → trello.trelloclient.TrelloClient
This function gets a Trello Client Object to connect to Trello.

Args: user_dict (dict): File path for the config file to be loaded

Returns: client (TrelloClient): Client object that has access to Trello.

polical.todo_generator.**get_uncompleted_cards**() → list
This function gets a all uncompleted cards from a board.

Returns: uncompleted_cards (List): List of strings from all uncompleted cards

polical.todo_generator.**update_done_taks**()
This functions gets new done tasks and updates to the cards on trello

polical.todo_generator.**update_new_taks**()
This function collects all new tasks from trello and concatenates to existing todo.txt file

polical.todo_generator.**write_todo_tasks**(*tasks_list: list*)
This function writes all tasks from a list.

Args: tasks_list (list): List containing tasks to be added to the todo.txt file

PYTHON MODULE INDEX

INDEX

L

M

O

P

S

T

U

W