Console Assistant

Technical description

Chapter 1

Console Assistant

Console Assistant is a `python 3.x`-based command-line program that allows you to manage your contacts and notes, sort files in a folder by category and delete file copies..

The assistant saves contacts and notes to the folder of the current user. For example, in `Windows` it's a folder `C:\Users\<User>\` with names

- `contacts.bin`
- `notes.bin`

respectively.

1.1 Install and Run Assistant

- Clone repository

```
https://github.com/sergiokapone/project-team-9_console_helper.git
```

- Create virtual environment in folder contains `setup.py`, in command line type

```
python -m venv .venv
```

or

```
py -m venv .venv
```

- Type `activate.cmd` in command line.
- To install a module in a virtual environment, type the `install.cmd` in console.
- To start the assistant, in console type:

```
assistant
```

The following packages are required to install the package:

```
prettytable==3.7.0
prompt-toolkit==3.0.0
Pygments==2.15.1
transliterate==1.10.2
```

Install requirements

```
pip install -r requirements.txt
```

1.2 Commands

1.2.1 Hello and welcome commands

- `help`: Displays a list of available commands and their descriptions.
- `hello`: Greets the user.
- `good bye`, `close`, `exit`: Exits the program with saving data.
- `.`: exits the program without messages with saving data.
- `*`: exits the program without messages without saving data.

1.2.2 Manage contacts commands

- `add contact`: Add a contact to the address book.
- `set phone:` Allows the user to set their phone number.
- `remove phone:` Removes the user phone number.
- `set email`: Allows the user to set their email address.
- `remove email:` Removes the user email.
- `set address`: Allows the user to set their address.
- `remove address:` Removes the user address.
- `set birthday`: Allows the user to set their birthday.
- `upcoming birthdays`: Shows the list of upcoming birthdays.
- `show contacts`: Shows the list of saved contacts.
- `show contact`: Shows the details of a specific contact.
- `search contact`: Searches for a specific contact by name.
- `remove contact`: Removes a contact from the list.
- `change name`: Change name of contact.
- `save`: save contacts to file.
- `load`: load contacts from file.

1.2.3 Currency exchange

- `currency`: Get Currency exchange.

1.2.4 Sorting Folder

- `sort folder`: Sort folder.

1.2.5 Manage notes commands

- `add note`: Adds a note.
- `show notes`: Shows the list of saved notes.
- `search notes`: Searches for a specific note by title or date.
- `remove note`: Removes a note from the list.
- `save notes`: Saves the notes to a file.
- `load notes`: Loads the notes from a file.
- `change note`: Edit note by index.
- `remove tag`: Removes Tag for specified note index.

1.3 Usage Examples

1.3.1 Sort folder

- `>> sort folder D:\MyGarbage\` — sort files in folder `D:\MyGarbage\`

1.3.2 Manage contacts

To manage contacts in your address book, You can type commands by following examples:

- `>> show contacts` — shows table of contacts in address book.
- `>> show contacts 3` displaying contacts in chunks of `3` items, `20` is by default
- `>> add contact Username` — add empty contact `Username` to the address book.
- `>> add contact Someone 03.05.1995 his_mail@i.ua` — add contact `Someone` to the address book with date of birthday `03.05.1995` and email `his_mail@i.ua`[1].
- `>> set phone Username 0935841245` — add phone[2] to contact `Username` in address book.
- `>> set birthday Username 12.12.1978` — add birthday[3] to contact `Username` in address book.
- `>> set email my_name@gmail.com` — add email[4] `my_name@gmail.com` to contact `Username` in address book.
- `>> upcoming birthdays 5` — shows contacts with upcoming birthdays within `5` days.
- `>> show contact Username` — show contact `Username` information.
- `>> search contact SearchQuery` — where `SearchQuery` some word or number for searching.
- `>> remove contact Username` — remove contact `Uresname`
- `>> remove phone Username 1` — remove user phone by index `1`.
- `>> remove email Username 2` — remove user email by index `2`.
- `>> change contact Username Bobo` — change contact name `Uresname` to new name `Bobo`.

1.3.3 Manage notes

To manage contacts in your address book, You can type commands by following examples:

- `>> show notes` — just show all notes.
- `>> show notes Rec` — show all notes with tag `Rec`.
- `>> show notes 3` — displaying notes in chunks of `3` items.
- `>> add note Tag Text of your note` — add note with `Tag`[5] and text `Text of your note`.
- `>> add tag 1 Mytag` — add tag `Mytag`[6] to note with index `1`.
- `>> remove tag 4 Tag` — removes `Tag` for note with index `4`.
- `>> change note 1 New text` — change note `1` with `New text`.
- `>> remove note 2` — remove note with index `2`.

[1] Order doesn't matter.
[2] If the contact is missing, it will be added automatically.
[3] If the contact is missing, it will be added automatically.
[4] If the contact is missing, it will be added automatically.
[5] When creating a note, you can assign only one single-word tag. You can add subsequent tags with `add tag` command.
[6] Tag cannot be a number.

1.3.4 Exit program

To exit the program just type following commands[7]:

- »> good bye
- »> close
- »> exit

Another possibility to exit without any messages is typing dot:

- »> .

Upon subsequent program entry, the data will be loaded automatically.

But if you type * the application exit without saving your data:

- »> *

[7] After executing the specified commands, the application will save your data automatically.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 console_assistant Namespace Reference

Namespaces

- namespace addressbook
- namespace CLI
- namespace colors
- namespace curreny
- namespace file_copies_deleter
- namespace filesorter
- namespace normaliser
- namespace notebook
- namespace serializer

6.2 console_assistant.addressbook Namespace Reference

Classes

- class AddressBook
- class Birthday
- class Email
- class Field
- class Phone

Variables

- namedtuple Record = namedtuple("Record", ["name", "birthday", "phones", "emails", "address"])

6.2.1 Variable Documentation

6.2.1.1 Record

```
namedtuple console_assistant.addressbook.Record = namedtuple("Record", ["name", "birthday",
"phones", "emails", "address"])
```

Definition at line 60 of file addressbook.py.

6.3 console_assistant.CLI Namespace Reference

Classes

- class Command
- class CommandCompleter
- class CommandExecutor
- class CommandParser
- class ConsoleView
- class InputReader
- class UserView

Functions

- def input_error (func)
- def hello (*args)
- def good_bye (*args)
- def save (*args)
- def load (*args)
- def parse_contact_params (string)
- def add_contact (*args)
- def remove_contact (*args)
- def remove_phone (*args)
- def remove_email (*args)
- def remove_address (*args)
- def upcoming_birthdays (*args)
- def change_name (*args)
- def search_contact (*args)
- def show_contact (*args)
- def build_contacts_table (contacts)
- def _get_phones_str (phones)
- def _get_emails_str (emails)
- def show_contacts (*args)
- def add_note (*args)
- def remove_note (*args)
- def add_tag (*args)
- def build_notes_table (notes, original_indices=False)
- def show_notes (*args)
- def search_notes (*args)
- def save_notes (*args)
- def change_note (*args)
- def remove_tag (*args)
- def load_notes (*args)
- def get_currency_table (CurrencyList currency_list)
- def get_currency (*args)
- def get_weather (*args)
- def help_commands (*args)
- def sort_folder_cli (*args)
- def cls (*args)
- def main ()

Variables

- dict COMMANDS
- AddressBook contacts = AddressBook()
- Notebook notebook = Notebook()
- str NOTES_FILE = "notes.bin"
- str CONTACT_FILE = "contacts.bin"
- f HELLO_MESSAGE = f"{N}Hello, I'm an assistant v1.0.0 {G}(c) Team-9, GoIT 2023.{N}\nType {Y}help{N} for more information.{N}"

6.3.1 Detailed Description

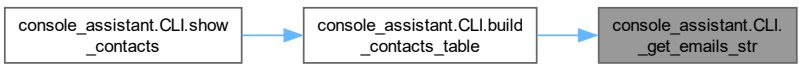
6.3.2 Function Documentation

6.3.2.1 _get_emails_str()

```
def console_assistant.CLI._get_emails_str (
              emails )  [protected]
```

Definition at line 424 of file CLI.py.

Here is the caller graph for this function:

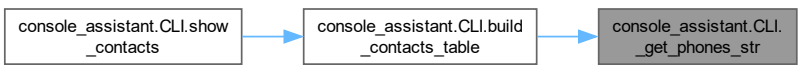


6.3.2.2 _get_phones_str()

```
def console_assistant.CLI._get_phones_str (
              phones )  [protected]
```

Definition at line 415 of file CLI.py.

Here is the caller graph for this function:

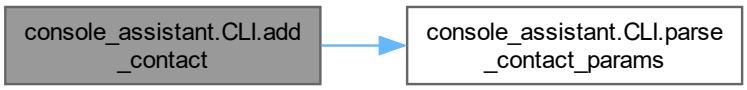


6.3.2.3 add_contact()

```
def console_assistant.CLI.add_contact (
            * args )
```

Definition at line 119 of file CLI.py.

Here is the call graph for this function:



6.3.2.4 add_note()

```
def console_assistant.CLI.add_note (
            * args )
```

Definition at line 461 of file CLI.py.

6.3.2.5 add_tag()

```
def console_assistant.CLI.add_tag (
            * args )
```

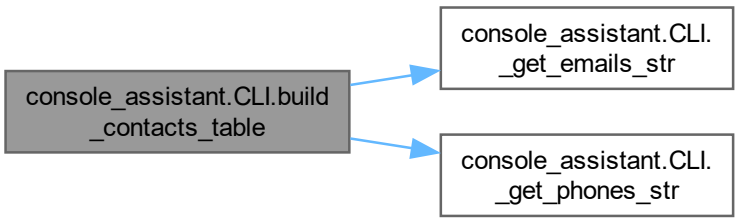
Definition at line 489 of file CLI.py.

6.3.2.6 build_contacts_table()

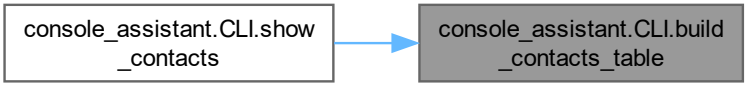
```
def console_assistant.CLI.build_contacts_table (
              contacts )
```

Definition at line 391 of file CLI.py.

Here is the call graph for this function:



Here is the caller graph for this function:

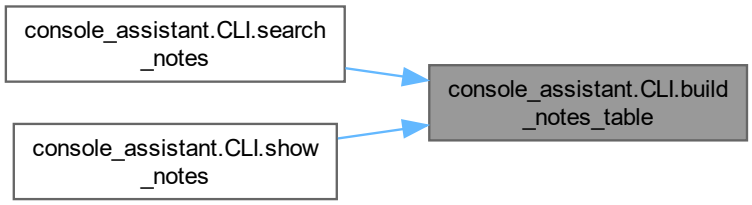


6.3.2.7 build_notes_table()

```
def console_assistant.CLI.build_notes_table (
              notes,
              original_indices = False )
```

Definition at line 509 of file CLI.py.

Here is the caller graph for this function:



6.3.2.8 change_name()

```
def console_assistant.CLI.change_name (
            * args )
```

Definition at line 349 of file CLI.py.

6.3.2.9 change_note()

```
def console_assistant.CLI.change_note (
            * args )
```

Definition at line 568 of file CLI.py.

6.3.2.10 cls()

```
def console_assistant.CLI.cls (
            * args )
```

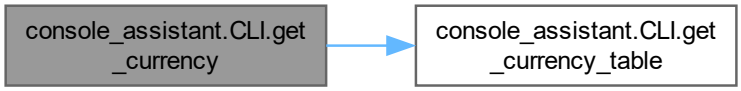
Definition at line 680 of file CLI.py.

6.3.2.11 get_currency()

```
def console_assistant.CLI.get_currency (
            * args )
```

Definition at line 625 of file CLI.py.

Here is the call graph for this function:

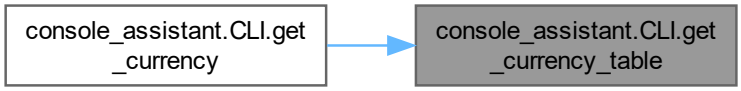


6.3.2.12 get_currency_table()

```
def console_assistant.CLI.get_currency_table (
            CurrencyList currency_list )
```

Definition at line 611 of file CLI.py.

Here is the caller graph for this function:



6.3.2.13 get_weather()

```
def console_assistant.CLI.get_weather (
            * args )
```

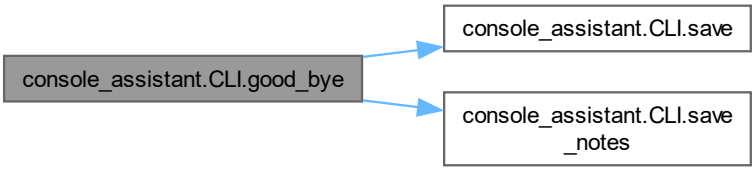
Definition at line 634 of file CLI.py.

6.3.2.14 good_bye()

```
def console_assistant.CLI.good_bye (
            * args )
```

Definition at line 65 of file CLI.py.

Here is the call graph for this function:



6.3.2.15 hello()

```
def console_assistant.CLI.hello (
            * args )
```

Definition at line 61 of file CLI.py.

6.3.2.16 help_commands()

```
def console_assistant.CLI.help_commands (
            * args )
```

.

Definition at line 657 of file CLI.py.

6.3.2.17 input_error()

```
def console_assistant.CLI.input_error (
             func )
```

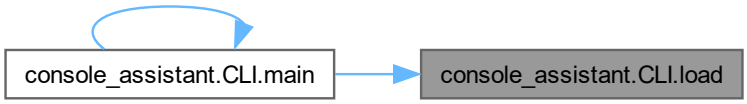
Definition at line 40 of file CLI.py.

6.3.2.18 load()

```
def console_assistant.CLI.load (
            * args )
```

Definition at line 85 of file CLI.py.

Here is the caller graph for this function:

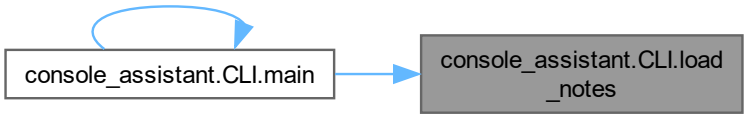


6.3.2.19 load_notes()

```
def console_assistant.CLI.load_notes (
            * args )
```

Definition at line 594 of file CLI.py.

Here is the caller graph for this function:

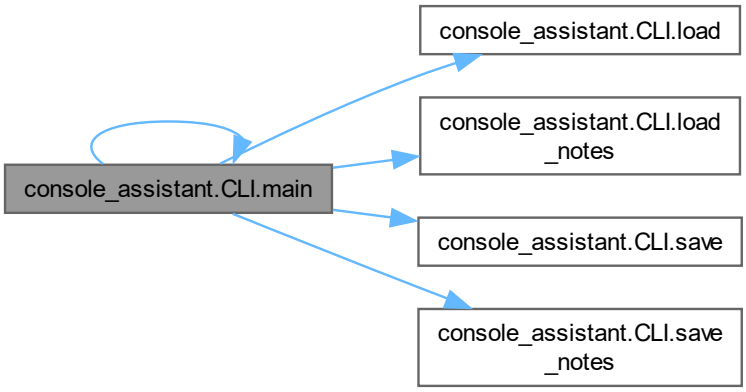


6.3.2.20 main()

```
def console_assistant.CLI.main ( )
```

Definition at line 829 of file CLI.py.

Here is the call graph for this function:



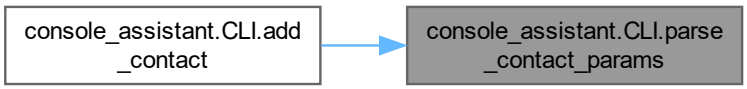
Here is the caller graph for this function:

6.3.2.21 parse_contact_params()

```
def console_assistant.CLI.parse_contact_params (
              string )
```

Definition at line 102 of file CLI.py.

Here is the caller graph for this function:



6.3.2.22 remove_address()

```
def console_assistant.CLI.remove_address (
            * args )
```

email .

Definition at line 295 of file CLI.py.

6.3.2.23 remove_contact()

```
def console_assistant.CLI.remove_contact (
            * args )
```

-handler .

Definition at line 155 of file CLI.py.

6.3.2.24 remove_email()

```
def console_assistant.CLI.remove_email (
            * args )
```

email .

Definition at line 247 of file CLI.py.

6.3.2.25 remove_note()

```
def console_assistant.CLI.remove_note (
            * args )
```

Definition at line 479 of file CLI.py.

6.3.2.26 remove_phone()

```
def console_assistant.CLI.remove_phone (
            * args )
```

.

Definition at line 200 of file CLI.py.

6.3.2.27 remove_tag()

```
def console_assistant.CLI.remove_tag (
            * args )
```

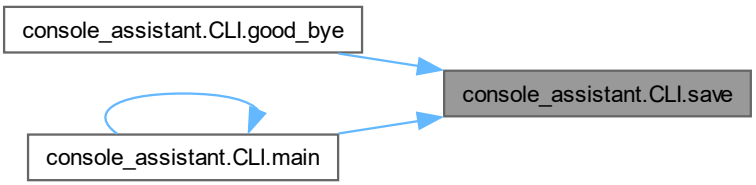
Definition at line 578 of file CLI.py.

6.3.2.28 save()

```
def console_assistant.CLI.save (
            * args )
```

Definition at line 74 of file CLI.py.

Here is the caller graph for this function:

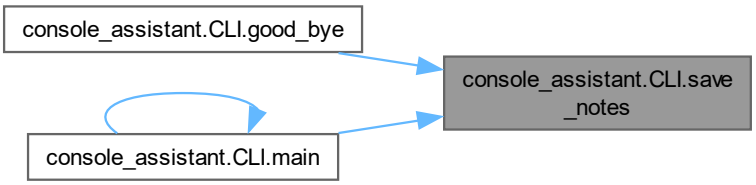


6.3.2.29 save_notes()

```
def console_assistant.CLI.save_notes (
            * args )
```

Definition at line 558 of file CLI.py.

Here is the caller graph for this function:



6.3.2.30 search_contact()

```
def console_assistant.CLI.search_contact (
            * args )
```

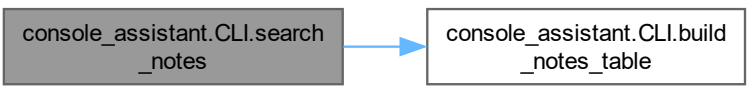
Definition at line 370 of file CLI.py.

6.3.2.31 search_notes()

```
def console_assistant.CLI.search_notes (
            * args )
```

Definition at line 548 of file CLI.py.

Here is the call graph for this function:



6.3.2.32 show_contact()

```
def console_assistant.CLI.show_contact (
            * args )
```

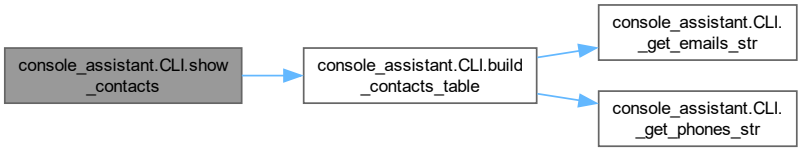
Definition at line 382 of file CLI.py.

6.3.2.33 show_contacts()

```
def console_assistant.CLI.show_contacts (
            * args )
```

Definition at line 434 of file CLI.py.

Here is the call graph for this function:

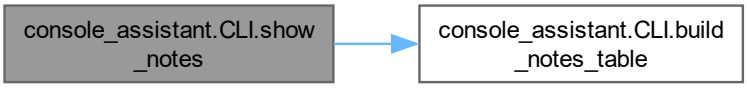


6.3.2.34 show_notes()

```
def console_assistant.CLI.show_notes (
            * args )
```

Definition at line 531 of file CLI.py.

Here is the call graph for this function:



6.3.2.35 sort_folder_cli()

```
def console_assistant.CLI.sort_folder_cli (
            * args )
```

Definition at line 674 of file CLI.py.

6.3.2.36 upcoming_birthdays()

```
def console_assistant.CLI.upcoming_birthdays (
            * args )
```

Definition at line 340 of file CLI.py.

6.3.3 Variable Documentation

6.3.3.1 COMMANDS

```
dict console_assistant.CLI.COMMANDS
```

Definition at line 695 of file CLI.py.

6.3.3.2 CONTACT_FILE

```
str console_assistant.CLI.CONTACT_FILE = "contacts.bin"
```

Definition at line 817 of file CLI.py.

6.3.3.3 contacts

```
AddressBook console_assistant.CLI.contacts = AddressBook()
```

Definition at line 812 of file CLI.py.

6.3.3.4 HELLO_MESSAGE

```
f console_assistant.CLI.HELLO_MESSAGE = f"{N}Hello, I'm an assistant v1.0.0 {G}(c) Team-9,
GoIT 2023.{N}\nType {Y}help{N} for more information.{N}"
```

Definition at line 818 of file CLI.py.

6.3.3.5 notebook

```
Notebook console_assistant.CLI.notebook = Notebook()
```

Definition at line 813 of file CLI.py.

6.3.3.6 NOTES_FILE

```
str console_assistant.CLI.NOTES_FILE = "notes.bin"
```

Definition at line 816 of file CLI.py.

6.4 console_assistant.colors Namespace Reference

Variables

- str G = "\033[1;92m"
- str B = "\033[1;96m"
- str P = "\033[4;95m"
- str R = "\033[1;91m"
- str N = "\033[0m"
- str Y = "\033[0;93m"
- str W = "\033[97m"

6.4.1 Variable Documentation

6.4.1.1 B

```
str console_assistant.colors.B = "\033[1;96m"
```

Definition at line 2 of file colors.py.

6.4.1.2 G

```
str console_assistant.colors.G = "\033[1;92m"
```

Definition at line 1 of file colors.py.

6.4.1.3 N

```
str console_assistant.colors.N = "\033[0m"
```

Definition at line 5 of file colors.py.

6.4.1.4 P

```
str console_assistant.colors.P = "\033[4;95m"
```

Definition at line 3 of file colors.py.

6.4.1.5 R

```
str console_assistant.colors.R = "\033[1;91m"
```

Definition at line 4 of file colors.py.

6.4.1.6 W

```
str console_assistant.colors.W = "\033[97m"
```

Definition at line 7 of file colors.py.

6.4.1.7 Y

```
str console_assistant.colors.Y = "\033[0;93m"
```

Definition at line 6 of file colors.py.

6.5 console_assistant.curreny Namespace Reference

Classes

- class CurrencyList

Functions

- def get_currency_table (CurrencyList currency_list)

Variables

- namedtuple Currency = namedtuple("Currency", ["name", "rate", "cc"])
- CurrencyList cur = CurrencyList()

6.5.1 Function Documentation

6.5.1.1 get_currency_table()

```
def console_assistant.curreny.get_currency_table (
            CurrencyList currency_list )
```

Definition at line 35 of file curreny.py.

6.5.2 Variable Documentation

6.5.2.1 cur

```
CurrencyList console_assistant.curreny.cur = CurrencyList()
```

Definition at line 47 of file curreny.py.

6.5.2.2 Currency

```
namedtuple console_assistant.curreny.Currency = namedtuple("Currency", ["name", "rate", "cc"])
```

Definition at line 6 of file curreny.py.

6.6 console_assistant.file_copies_deleter Namespace Reference

Functions

- def count_files (abs_path, file_format="")
- def get_hash (path_to_file)
- def count_copies (dir_path, file_format="")
- def delete_files (hash_dictionary)
- def delete_empty_folders (path)
- def copies_deleter (root)

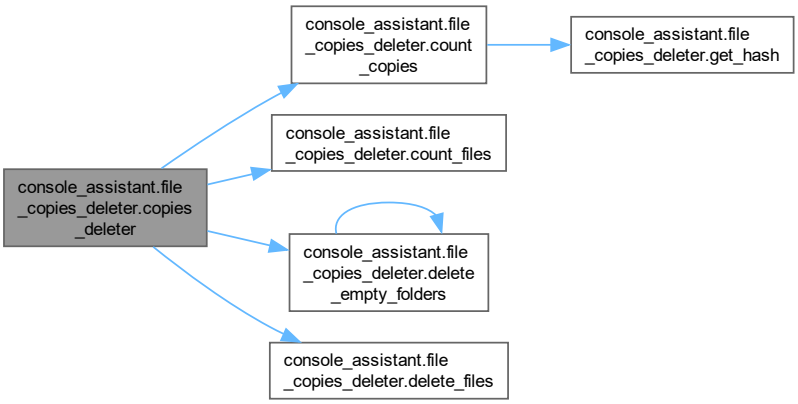
6.6.1 Function Documentation

6.6.1.1 copies_deleter()

```
def console_assistant.file_copies_deleter.copies_deleter (
              root )
```

Definition at line 81 of file file_copies_deleter.py.

Here is the call graph for this function:

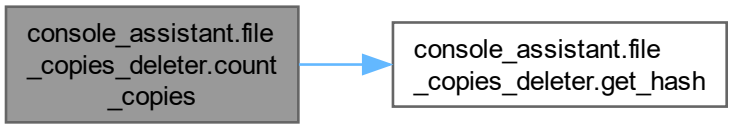


6.6.1.2 count_copies()

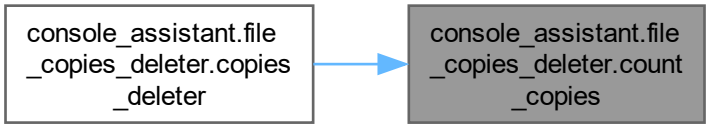
```
def console_assistant.file_copies_deleter.count_copies (
              dir_path,
              file_format = "" )
```

Definition at line 40 of file file_copies_deleter.py.

Here is the call graph for this function:



Here is the caller graph for this function:

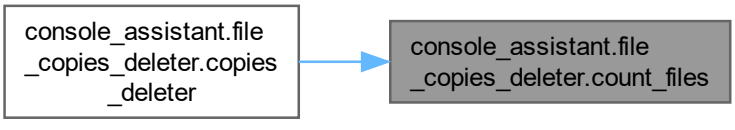


6.6.1.3 count_files()

```
def console_assistant.file_copies_deleter.count_files (
              abs_path,
              file_format = "" )
```

Definition at line 20 of file file_copies_deleter.py.

Here is the caller graph for this function:

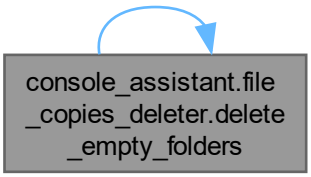


6.6.1.4 delete_empty_folders()

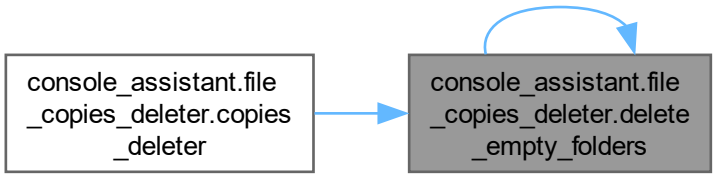
```
def console_assistant.file_copies_deleter.delete_empty_folders (
              path )
```

Definition at line 72 of file file_copies_deleter.py.

Here is the call graph for this function:



Here is the caller graph for this function:

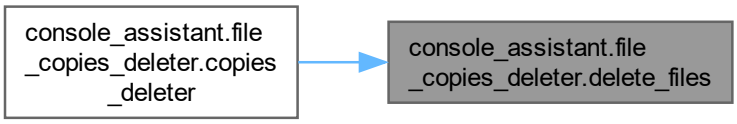


6.6.1.5 delete_files()

```
def console_assistant.file_copies_deleter.delete_files (
              hash_dictionary )
```

Definition at line 61 of file file_copies_deleter.py.

Here is the caller graph for this function:

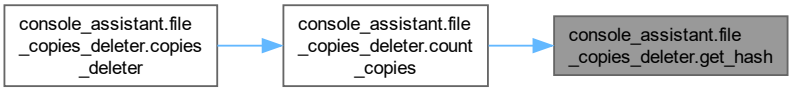


6.6.1.6 get_hash()

```
def console_assistant.file_copies_deleter.get_hash (
              path_to_file )
```

Definition at line 32 of file file_copies_deleter.py.

Here is the caller graph for this function:



6.7 console_assistant.filesorter Namespace Reference

Functions

- def get_file_cathegory (str file)
- def normalise_file_name (str file)
- def create_folders (root)
- def organize_files (path, level=0, known_exts=set(), unknown_exts=set(), categories=set())
- def unpack (archive_path, path_to_unpack)
- def remove_empty (path)
- def known_exts (root)
- def sort_folder (root)

Variables

- dict EXT_FOLDER

6.7.1 Function Documentation

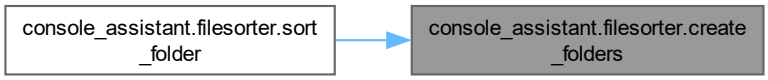
6.7.1.1 create_folders()

```
def console_assistant.filesorter.create_folders (
              root )
```

root

Definition at line 46 of file filesorter.py.

Here is the caller graph for this function:



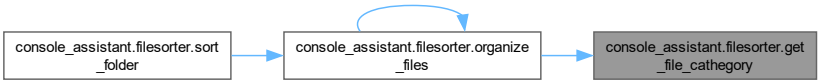
6.7.1.2 get_file_cathegory()

```
def console_assistant.filesorter.get_file_cathegory (
             str file )
```

.

Definition at line 22 of file filesorter.py.

Here is the caller graph for this function:

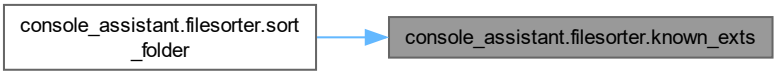


6.7.1.3 known_exts()

```
def console_assistant.filesorter.known_exts (
              root )
```

Definition at line 112 of file filesorter.py.

Here is the caller graph for this function:



6.7.1.4 normalise_file_name()

```
def console_assistant.filesorter.normalise_file_name (
             str file )


      normalise.
```

Definition at line 34 of file filesorter.py.

Here is the caller graph for this function:

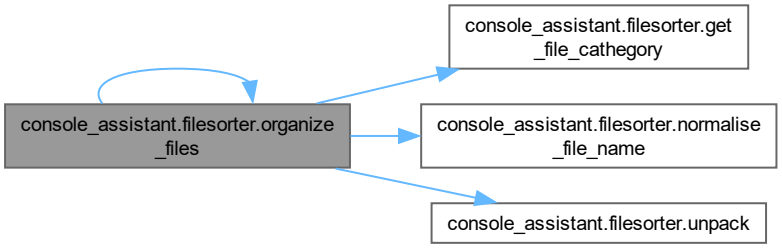
6.7.1.5 organize_files()

```
def console_assistant.filesorter.organize_files (
              path,
              level = 0,
              known_exts = set(),
              unknown_exts = set(),
              categories = set() )
```

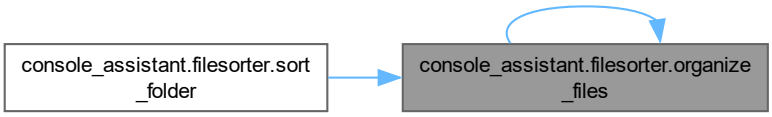
.

Definition at line 53 of file filesorter.py.

Here is the call graph for this function:



Here is the caller graph for this function:



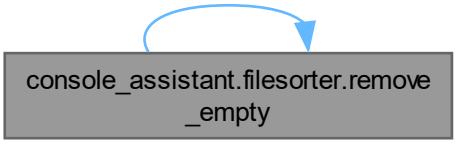
6.7.1.6 remove_empty()

```
def console_assistant.filesorter.remove_empty (
              path )
```

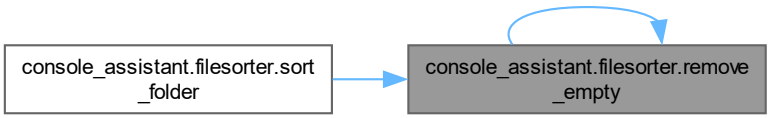
.

Definition at line 96 of file filesorter.py.

Here is the call graph for this function:



Here is the caller graph for this function:

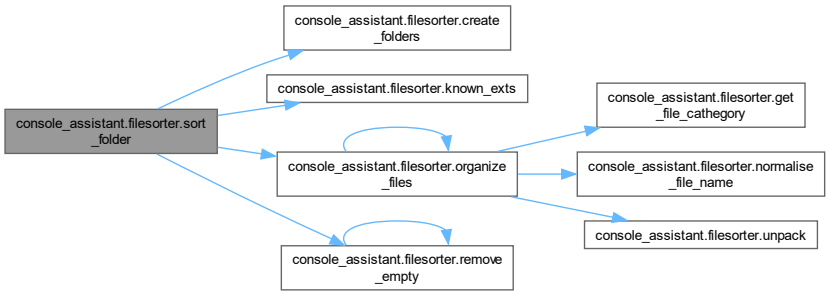


6.7.1.7 sort_folder()

```
def console_assistant.filesorter.sort_folder (
              root )
```

Definition at line 125 of file filesorter.py.

Here is the call graph for this function:



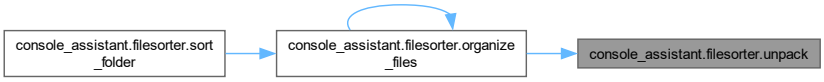
6.7.1.8 unpack()

```
def console_assistant.filesorter.unpack (
              archive_path,
              path_to_unpack )
```

.

Definition at line 87 of file filesorter.py.

Here is the caller graph for this function:



6.7.2 Variable Documentation

6.7.2.1 EXT_FOLDER

```
dict console_assistant.filesorter.EXT_FOLDER
```

Initial value:

```
00001 =  {
00002     ("mp3", "ogg", "waw", "amr"): "audio",
00003     ("avi", "mp4", "mov", "mkv", "flv"): "video",
00004     ("jpeg", "png", "jpg", "svg"): "images",
00005     ("doc", "docx", "txt", "xlsx", "xls", "pptx"): "documents",
00006     ("djvu", "djv", "pdf", "tiff"): "books",
00007     ("zip", "gz", "tar", "7z"): "archives",
00008 }
```

Definition at line 10 of file filesorter.py.

6.8 console_assistant.normaliser Namespace Reference

Functions

- def normalise (name)

6.8.1 Function Documentation

6.8.1.1 normalise()

```
def console_assistant.normaliser.normalise (
              name )
```

Definition at line 5 of file normaliser.py.

6.9 console_assistant.notebook Namespace Reference

Classes

- class Notebook

Variables

- namedtuple Note = namedtuple("Note", ["tags", "date", "text"])

6.9.1 Variable Documentation

6.9.1.1 Note

```
namedtuple console_assistant.notebook.Note = namedtuple("Note", ["tags", "date", "text"])
```

Definition at line 4 of file notebook.py.

6.10 console_assistant.serializer Namespace Reference

Classes

- class PickleStorage
- class Storage

6.10.1 Detailed Description

6.11 setup Namespace Reference

Variables

- name
- version
- description
- author
- license
- url
- include_package_data
- packages
- entry_points
- install_requires
- package_data

6.11.1 Variable Documentation

6.11.1.1 author

```
setup.author
```

Definition at line 7 of file setup.py.

6.11.1.2 description

```
setup.description
```

Definition at line 6 of file setup.py.

6.11.1.3 entry_points

```
setup.entry_points
```

Definition at line 12 of file setup.py.

6.11.1.4 include_package_data

```
setup.include_package_data
```

Definition at line 10 of file setup.py.

6.11.1.5 install_requires

```
setup.install_requires
```

Definition at line 13 of file setup.py.

6.11.1.6 license

```
setup.license
```

Definition at line 8 of file setup.py.

6.11.1.7 name

```
setup.name
```

Definition at line 4 of file setup.py.

6.11.1.8 package_data

```
setup.package_data
```

Definition at line 21 of file setup.py.

6.11.1.9 packages

```
setup.packages
```

Definition at line 11 of file setup.py.

6.11.1.10 url

```
setup.url
```

Definition at line 9 of file setup.py.

6.11.1.11 version

```
setup.version
```

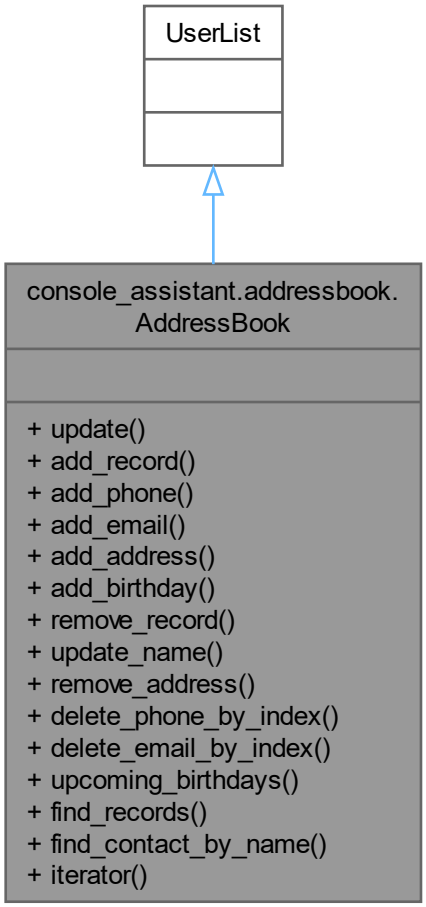
Definition at line 5 of file setup.py.

Chapter 7

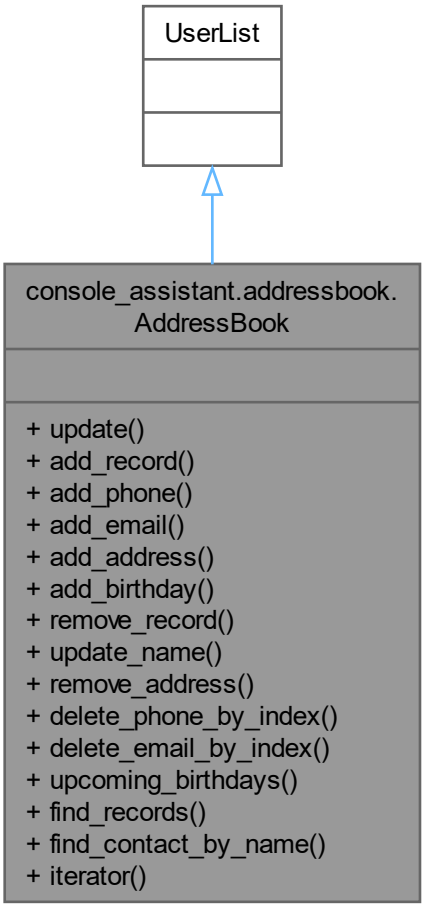
Class Documentation

7.1 console_assistant.addressbook.AddressBook Class Reference

Inheritance diagram for console_assistant.addressbook.AddressBook:



Collaboration diagram for console_assistant.addressbook.AddressBook:



Public Member Functions

- def update (self, records)
- def add_record (self, name, birthday=None, phones=None, emails=None, address=None)
- def add_phone (self, name, phone)
- def add_email (self, name, email)
- def add_address (self, name, address)
- def add_birthday (self, name, birthday)
- def remove_record (self, name)
- def update_name (self, name, new_name)
- def remove_address (self, name)
- def delete_phone_by_index (self, name, phone_index)
- def delete_email_by_index (self, name, email_index)
- def upcoming_birthdays (self, days)
- def find_records (self, search_term)
- def find_contact_by_name (self, name)
- def iterator (self, int n=10)

7.1.1 Detailed Description

Definition at line 63 of file addressbook.py.

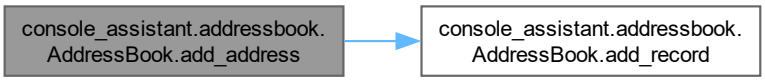
7.1.2 Member Function Documentation

7.1.2.1 add_address()

```
def console_assistant.addressbook.AddressBook.add_address (
              self,
              name,
              address )
```

Definition at line 112 of file addressbook.py.

Here is the call graph for this function:

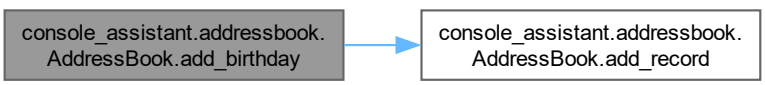


7.1.2.2 add_birthday()

```
def console_assistant.addressbook.AddressBook.add_birthday (
              self,
              name,
              birthday )
```

Definition at line 120 of file addressbook.py.

Here is the call graph for this function:

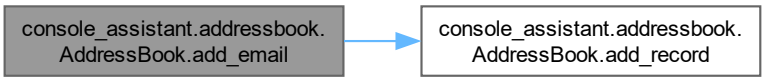


7.1.2.3 add_email()

```
def console_assistant.addressbook.AddressBook.add_email (
                self,
                name,
                email )
```

Definition at line 105 of file addressbook.py.

Here is the call graph for this function:

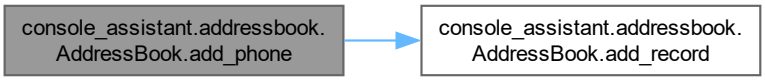


7.1.2.4 add_phone()

```
def console_assistant.addressbook.AddressBook.add_phone (
                self,
                name,
                phone )
```

Definition at line 98 of file addressbook.py.

Here is the call graph for this function:

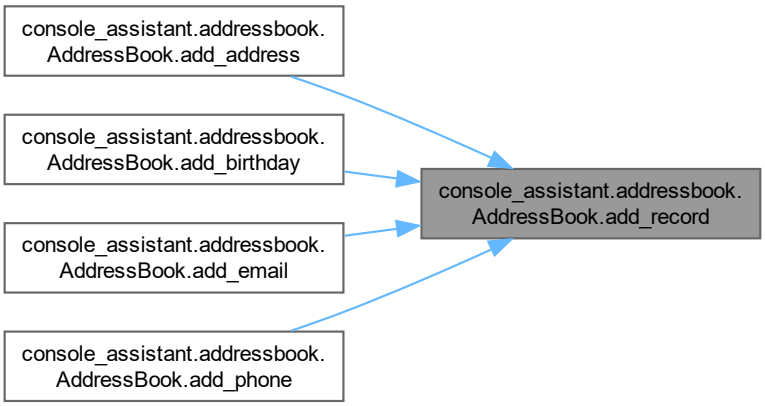


7.1.2.5 add_record()

```
def console_assistant.addressbook.AddressBook.add_record (
              self,
              name,
              birthday = None,
              phones = None,
              emails = None,
              address = None )
```

Definition at line 68 of file addressbook.py.

Here is the caller graph for this function:



7.1.2.6 delete_email_by_index()

```
def console_assistant.addressbook.AddressBook.delete_email_by_index (
              self,
              name,
              email_index )
```

Definition at line 157 of file addressbook.py.

7.1.2.7 delete_phone_by_index()

```
def console_assistant.addressbook.AddressBook.delete_phone_by_index (
              self,
              name,
              phone_index )
```

Definition at line 150 of file addressbook.py.

7.1.2.8 find_contact_by_name()

```
def console_assistant.addressbook.AddressBook.find_contact_by_name (
              self,
              name )
```

Definition at line 200 of file addressbook.py.

7.1.2.9 find_records()

```
def console_assistant.addressbook.AddressBook.find_records (
              self,
              search_term )
```

Definition at line 185 of file addressbook.py.

7.1.2.10 iterator()

```
def console_assistant.addressbook.AddressBook.iterator (
              self,
             int  n = 10 )


         n-.
```

Definition at line 206 of file addressbook.py.

7.1.2.11 remove_address()

```
def console_assistant.addressbook.AddressBook.remove_address (
              self,
              name )
```

Definition at line 142 of file addressbook.py.

7.1.2.12 remove_record()

```
def console_assistant.addressbook.AddressBook.remove_record (
              self,
              name )
```

Definition at line 128 of file addressbook.py.

7.1.2.13 upcoming_birthdays()

```
def console_assistant.addressbook.AddressBook.upcoming_birthdays (
              self,
              days )


         days
```

Definition at line 164 of file addressbook.py.

7.1.2.14 update()

```
def console_assistant.addressbook.AddressBook.update (
              self,
              records )
```

Definition at line 64 of file addressbook.py.

7.1.2.15 update_name()

```
def console_assistant.addressbook.AddressBook.update_name (
              self,
              name,
              new_name )
```

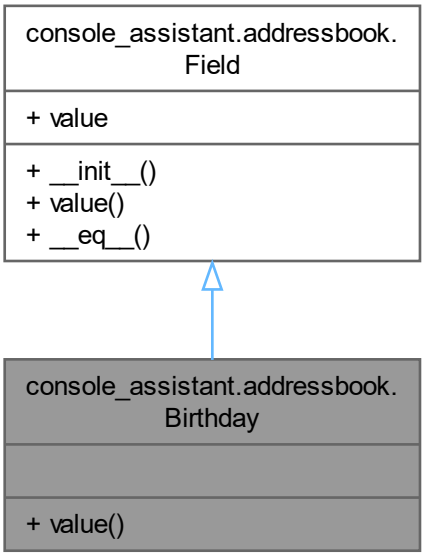
Definition at line 135 of file addressbook.py.

The documentation for this class was generated from the following file:

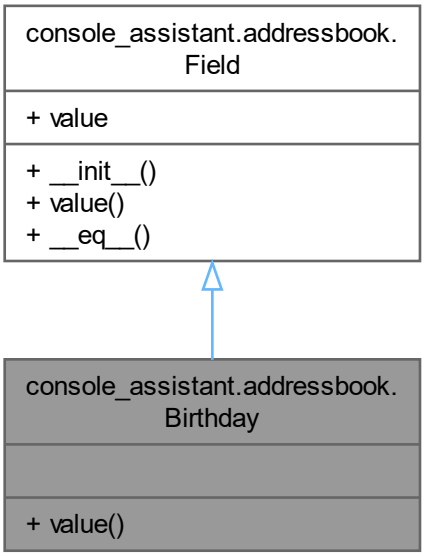
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/addressbook.py

7.2 console_assistant.addressbook.Birthday Class Reference

Inheritance diagram for console_assistant.addressbook.Birthday:



Collaboration diagram for console_assistant.addressbook.Birthday:



Public Member Functions

- def value (self, value)

Public Member Functions inherited from console_assistant.addressbook.Field

- def __init__ (self, str value)
- def value (self)
- def __eq__ (self, other)

Additional Inherited Members

Public Attributes inherited from console_assistant.addressbook.Field

- value

7.2.1 Detailed Description

--- ' .

Definition at line 36 of file addressbook.py.

7.2.2 Member Function Documentation

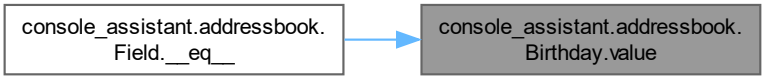
7.2.2.1 value()

```
def console_assistant.addressbook.Birthday.value (
              self,
              value )
```

Reimplemented from console_assistant.addressbook.Field.

Definition at line 40 of file addressbook.py.

Here is the caller graph for this function:

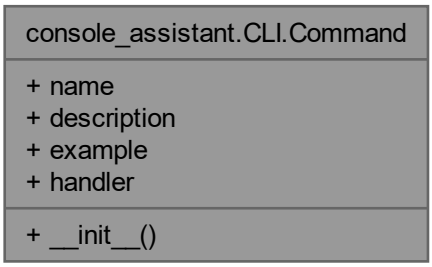


The documentation for this class was generated from the following file:

- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/addressbook.py

7.3 console_assistant.CLI.Command Class Reference

Collaboration diagram for console_assistant.CLI.Command:



Public Member Functions

- def __init__ (self, name, handler, description=None, example=None)

Public Attributes

- name
- description
- example
- handler

7.3.1 Detailed Description

Definition at line 687 of file CLI.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def console_assistant.CLI.Command.__init__ (
              self,
              name,
              handler,
              description = None,
              example = None )
```

Definition at line 688 of file CLI.py.

7.3.3 Member Data Documentation

7.3.3.1 description

`console_assistant.CLI.Command.description`

Definition at line 690 of file CLI.py.

7.3.3.2 example

`console_assistant.CLI.Command.example`

Definition at line 691 of file CLI.py.

7.3.3.3 handler

`console_assistant.CLI.Command.handler`

Definition at line 692 of file CLI.py.

7.3.3.4 name

`console_assistant.CLI.Command.name`

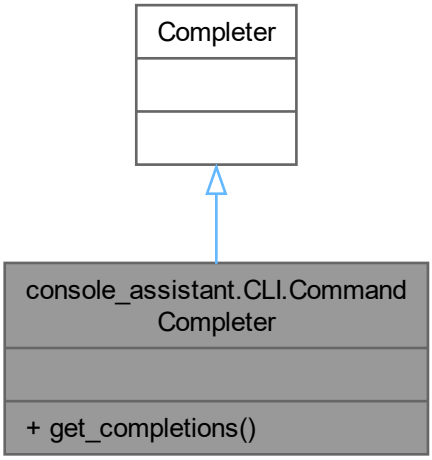
Definition at line 689 of file CLI.py.

The documentation for this class was generated from the following file:

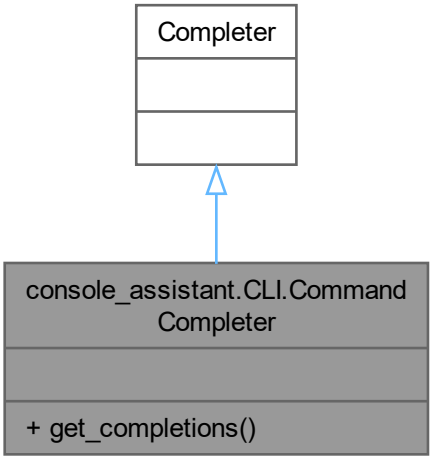
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.4 console_assistant.CLI.CommandCompleter Class Reference

Inheritance diagram for console_assistant.CLI.CommandCompleter:



Collaboration diagram for console_assistant.CLI.CommandCompleter:



Public Member Functions

- def get_completions (self, document, complete_event)

7.4.1 Detailed Description

Definition at line 739 of file CLI.py.

7.4.2 Member Function Documentation

7.4.2.1 get_completions()

```
def console_assistant.CLI.CommandCompleter.get_completions (
              self,
              document,
              complete_event )
```

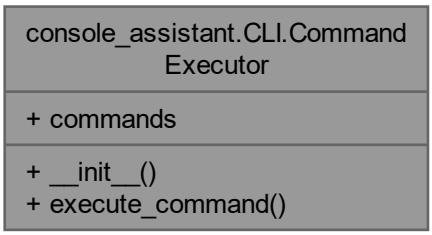
Definition at line 740 of file CLI.py.

The documentation for this class was generated from the following file:

- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.5 console_assistant.CLI.CommandExecutor Class Reference

Collaboration diagram for console_assistant.CLI.CommandExecutor:



Public Member Functions

- def __init__ (self, commands)
- def execute_command (self, command, ∗args)

Public Attributes

- commands

7.5.1 Detailed Description

Definition at line 776 of file CLI.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
def console_assistant.CLI.CommandExecutor.__init__ (
              self,
              commands )
```

Definition at line 777 of file CLI.py.

7.5.3 Member Function Documentation

7.5.3.1 execute_command()

```
def console_assistant.CLI.CommandExecutor.execute_command (
              self,
              command,
             * args )
```

Definition at line 780 of file CLI.py.

7.5.4 Member Data Documentation

7.5.4.1 commands

```
console_assistant.CLI.CommandExecutor.commands
```

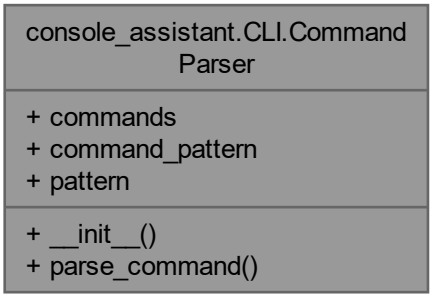
Definition at line 778 of file CLI.py.

The documentation for this class was generated from the following file:

- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.6 console_assistant.CLI.CommandParser Class Reference

Collaboration diagram for console_assistant.CLI.CommandParser:



Public Member Functions

- def __init__ (self, commands)
- def parse_command (self, command)

Public Attributes

- commands
- command_pattern
- pattern

7.6.1 Detailed Description

Definition at line 748 of file CLI.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def console_assistant.CLI.CommandParser.__init__ (
              self,
              commands )
```

Definition at line 749 of file CLI.py.

7.6.3 Member Function Documentation

7.6.3.1 parse_command()

```
def console_assistant.CLI.CommandParser.parse_command (
              self,
              command )
```

Definition at line 759 of file CLI.py.

7.6.4 Member Data Documentation

7.6.4.1 command_pattern

```
console_assistant.CLI.CommandParser.command_pattern
```

Definition at line 751 of file CLI.py.

7.6.4.2 commands

```
console_assistant.CLI.CommandParser.commands
```

Definition at line 750 of file CLI.py.

7.6.4.3 pattern

```
console_assistant.CLI.CommandParser.pattern
```

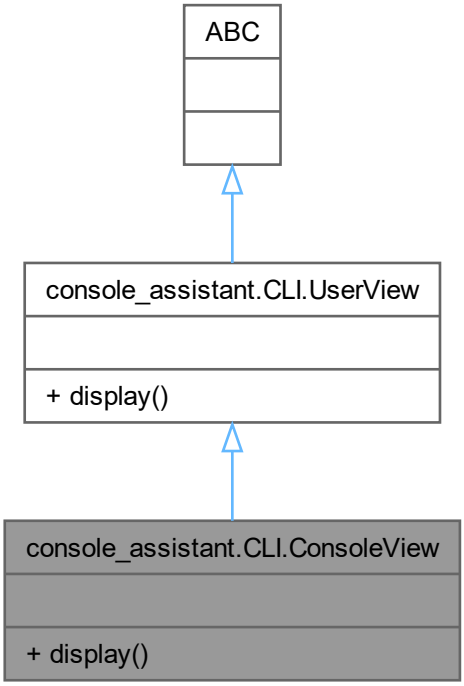
Definition at line 752 of file CLI.py.

The documentation for this class was generated from the following file:

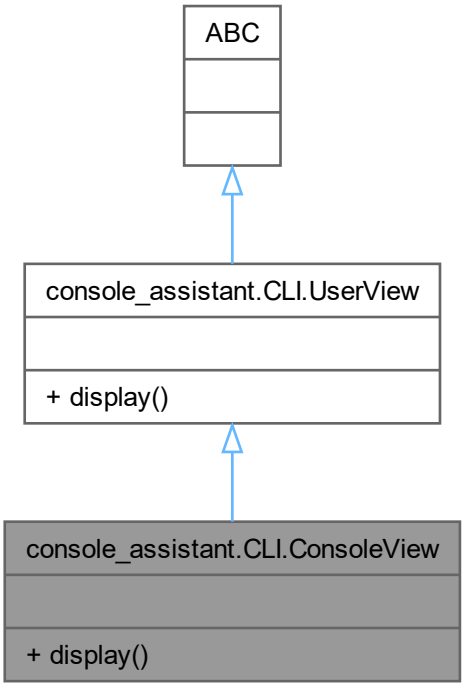
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.7 console_assistant.CLI.ConsoleView Class Reference

Inheritance diagram for console_assistant.CLI.ConsoleView:



Collaboration diagram for console_assistant.CLI.ConsoleView:



Public Member Functions

- def display (self, data)
- def display (self, data)

7.7.1 Detailed Description

Definition at line 824 of file CLI.py.

7.7.2 Member Function Documentation

7.7.2.1 display()

```
def console_assistant.CLI.ConsoleView.display (
              self,
              data )
```

Reimplemented from console_assistant.CLI.UserView.

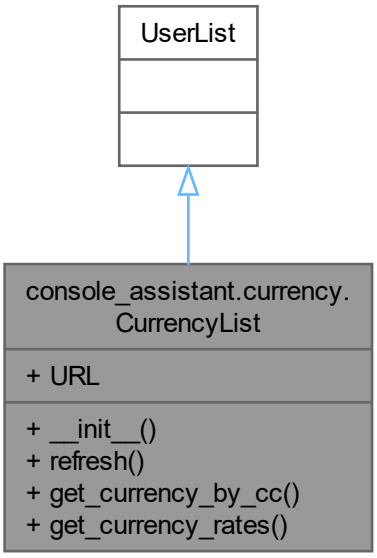
Definition at line 825 of file CLI.py.

The documentation for this class was generated from the following file:

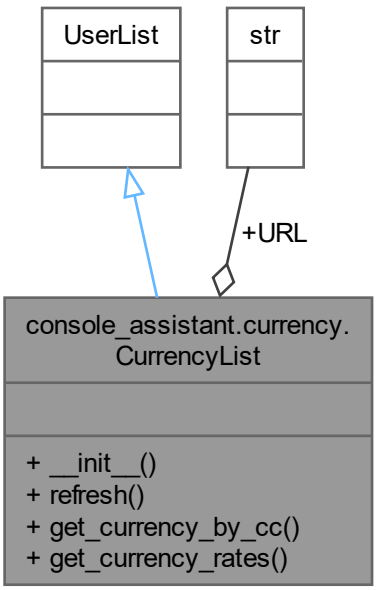
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.8 console_assistant.curreny.CurrencyList Class Reference

Inheritance diagram for console_assistant.curreny.CurrencyList:



Collaboration diagram for console_assistant.curreny.CurrencyList:



Public Member Functions

- def __init__ (self)
- def refresh (self)
- Currency get_currency_by_cc (self, str cc)
- def get_currency_rates (self)

Static Public Attributes

- str URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchangenew?json"

7.8.1 Detailed Description

Definition at line 9 of file curreny.py.

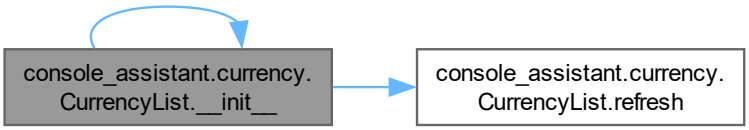
7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

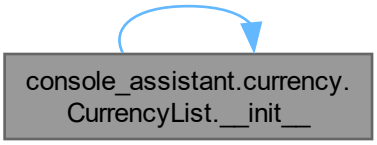
```
def console_assistant.curreny.CurrencyList.__init__ (
              self )
```

Definition at line 12 of file curreny.py.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3 Member Function Documentation

7.8.3.1 get_currency_by_cc()

```
Currency console_assistant.curreny.CurrencyList.get_currency_by_cc (
              self,
             str cc )
```

Definition at line 24 of file curreny.py.

7.8.3.2 get_currency_rates()

```
def console_assistant.curreny.CurrencyList.get_currency_rates (
              self )
```

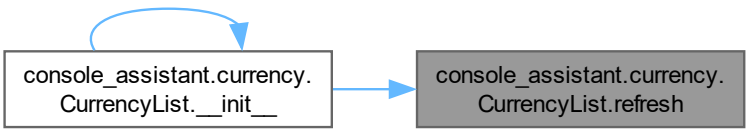
Definition at line 29 of file curreny.py.

7.8.3.3 refresh()

```
def console_assistant.curreny.CurrencyList.refresh (
              self )
```

Definition at line 16 of file curreny.py.

Here is the caller graph for this function:



7.8.4 Member Data Documentation

7.8.4.1 URL

```
str console_assistant.curreny.CurrencyList.URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange
[static]
```

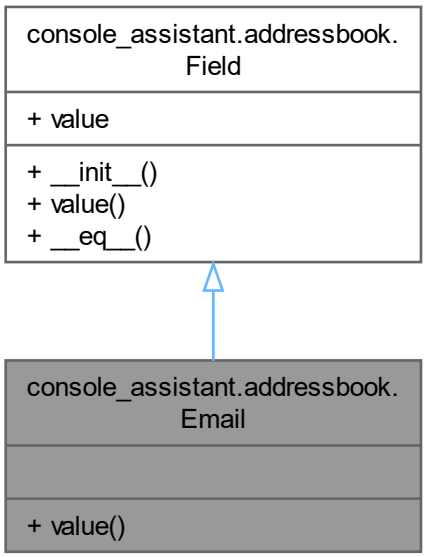
Definition at line 10 of file curreny.py.

The documentation for this class was generated from the following file:

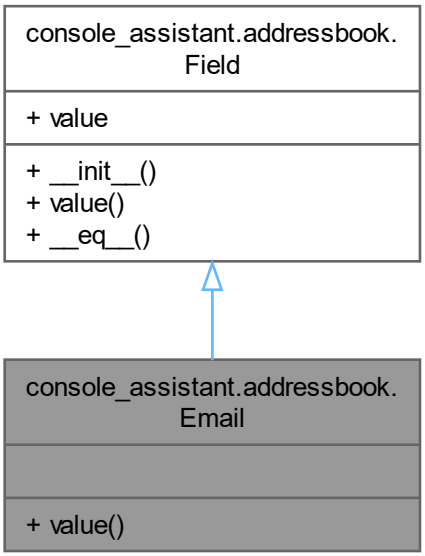
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/curreny.py

7.9 console_assistant.addressbook.Email Class Reference

Inheritance diagram for console_assistant.addressbook.Email:



Collaboration diagram for console_assistant.addressbook.Email:



Public Member Functions

- def value (self, value)

Public Member Functions inherited from console_assistant.addressbook.Field

- def __init__ (self, str value)
- def value (self)
- def __eq__ (self, other)

Additional Inherited Members

Public Attributes inherited from console_assistant.addressbook.Field

- value

7.9.1 Detailed Description

```
--- '   email
```

Definition at line 50 of file addressbook.py.

7.9.2 Member Function Documentation

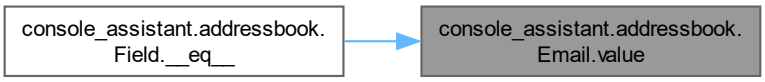
7.9.2.1 value()

```
def console_assistant.addressbook.Email.value (
              self,
              value )
```

Reimplemented from console_assistant.addressbook.Field.

Definition at line 54 of file addressbook.py.

Here is the caller graph for this function:

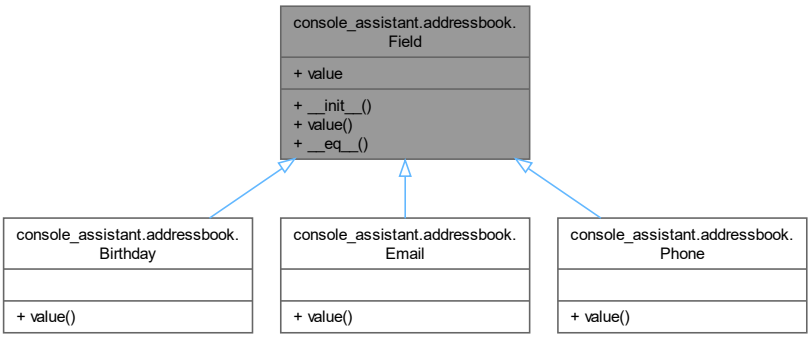


The documentation for this class was generated from the following file:

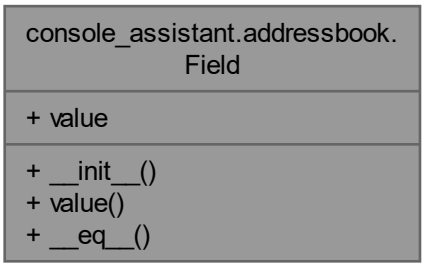
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/addressbook.py

7.10 console_assistant.addressbook.Field Class Reference

Inheritance diagram for console_assistant.addressbook.Field:



Collaboration diagram for console_assistant.addressbook.Field:



Public Member Functions

- def __init__ (self, str value)
- def value (self)
- def __eq__ (self, other)

Public Attributes

- value

7.10.1 Detailed Description

, ,
.

Definition at line 6 of file addressbook.py.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

```
def console_assistant.addressbook.Field.__init__ (
              self,
             str value )
```

Definition at line 10 of file addressbook.py.

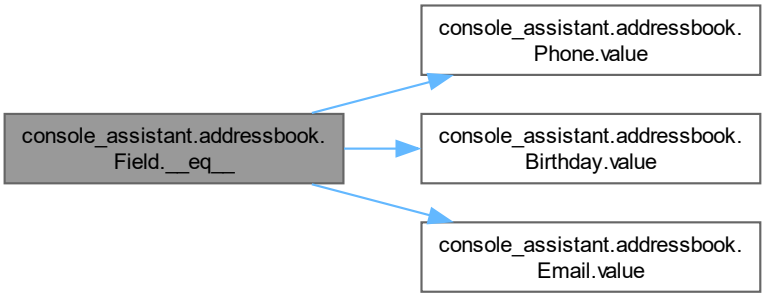
7.10.3 Member Function Documentation

7.10.3.1 __eq__()

```
def console_assistant.addressbook.Field.__eq__ (
              self,
              other )
```

Definition at line 18 of file addressbook.py.

Here is the call graph for this function:



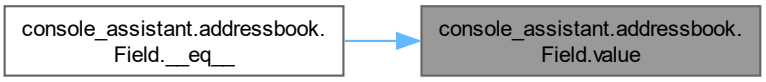
7.10.3.2 value()

```
def console_assistant.addressbook.Field.value (
              self )
```

Reimplemented in console_assistant.addressbook.Phone, console_assistant.addressbook.Birthday, and console_assistant.addressbook.Email.

Definition at line 15 of file addressbook.py.

Here is the caller graph for this function:



7.10.4 Member Data Documentation

7.10.4.1 value

`console_assistant.addressbook.Field.value`

Reimplemented in console_assistant.addressbook.Phone, console_assistant.addressbook.Birthday, and console_assistant.addressbook.Email.

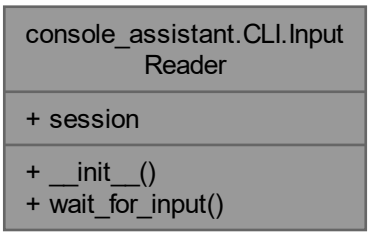
Definition at line 12 of file addressbook.py.

The documentation for this class was generated from the following file:

- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/addressbook.py

7.11 console_assistant.CLI.InputReader Class Reference

Collaboration diagram for console_assistant.CLI.InputReader:



Public Member Functions

- def __init__ (self)
- def wait_for_input (self)

Public Attributes

- session

7.11.1 Detailed Description

Definition at line 796 of file CLI.py.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 __init__()

```
def console_assistant.CLI.InputReader.__init__ (
              self )
```

Definition at line 797 of file CLI.py.

7.11.3 Member Function Documentation

7.11.3.1 wait_for_input()

```
def console_assistant.CLI.InputReader.wait_for_input (
              self )
```

Definition at line 800 of file CLI.py.

7.11.4 Member Data Documentation

7.11.4.1 session

```
console_assistant.CLI.InputReader.session
```

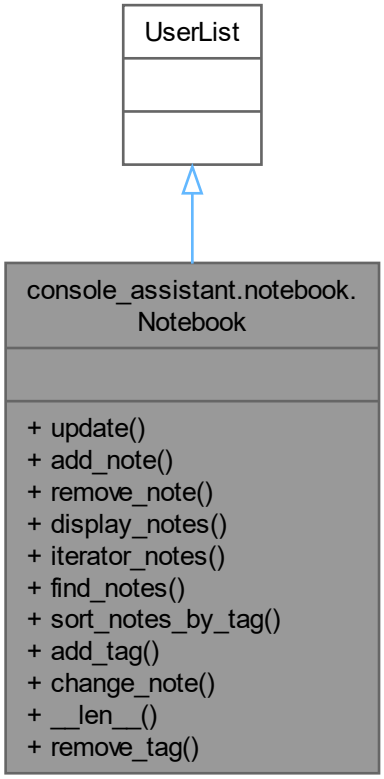
Definition at line 798 of file CLI.py.

The documentation for this class was generated from the following file:

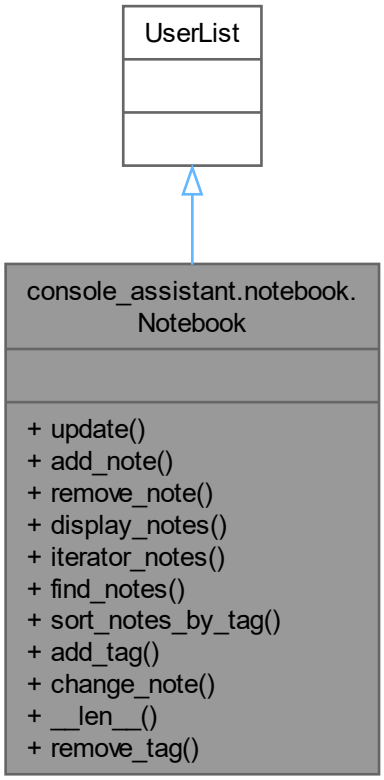
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

7.12 console_assistant.notebook.Notebook Class Reference

Inheritance diagram for console_assistant.notebook.Notebook:



Collaboration diagram for console_assistant.notebook.Notebook:



Public Member Functions

- def update (self, notes)
- def add_note (self, tags, note_text)
- def remove_note (self, index)
- def display_notes (self, tag=None, original_indices=False)
- def iterator_notes (self, int n=10)
- def find_notes (self, search_term)
- def sort_notes_by_tag (self)
- def add_tag (self, index, tag)
- def change_note (self, index, new_text)
- def __len__ (self)
- def remove_tag (self, index, tag)

7.12.1 Detailed Description

Definition at line 7 of file notebook.py.

7.12.2 Member Function Documentation

7.12.2.1 __len__()

```
def console_assistant.notebook.Notebook.__len__ (
              self )
```

Definition at line 71 of file notebook.py.

7.12.2.2 add_note()

```
def console_assistant.notebook.Notebook.add_note (
              self,
              tags,
              note_text )
```

Definition at line 13 of file notebook.py.

7.12.2.3 add_tag()

```
def console_assistant.notebook.Notebook.add_tag (
              self,
              index,
              tag )
```

.

Definition at line 57 of file notebook.py.

7.12.2.4 change_note()

```
def console_assistant.notebook.Notebook.change_note (
              self,
              index,
              new_text )
```

Definition at line 66 of file notebook.py.

7.12.2.5 display_notes()

```
def console_assistant.notebook.Notebook.display_notes (
              self,
              tag = None,
              original_indices = False )

 ,    ,     original_indices=True
```

Definition at line 23 of file notebook.py.

7.12.2.6 find_notes()

```
def console_assistant.notebook.Notebook.find_notes (
              self,
              search_term )
```

Definition at line 44 of file notebook.py.

7.12.2.7 iterator_notes()

```
def console_assistant.notebook.Notebook.iterator_notes (
              self,
             int  n = 10 )

         n-.
```

Definition at line 34 of file notebook.py.

7.12.2.8 remove_note()

```
def console_assistant.notebook.Notebook.remove_note (
              self,
              index )
```

Definition at line 19 of file notebook.py.

7.12.2.9 remove_tag()

```
def console_assistant.notebook.Notebook.remove_tag (
              self,
              index,
              tag )
```

Definition at line 74 of file notebook.py.

7.12.2.10 sort_notes_by_tag()

```
def console_assistant.notebook.Notebook.sort_notes_by_tag (
              self )
```

Definition at line 53 of file notebook.py.

7.12.2.11 update()

```
def console_assistant.notebook.Notebook.update (
              self,
              notes )
```

.

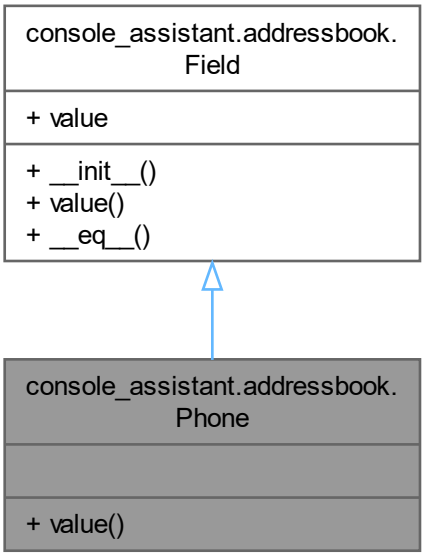
Definition at line 8 of file notebook.py.

The documentation for this class was generated from the following file:

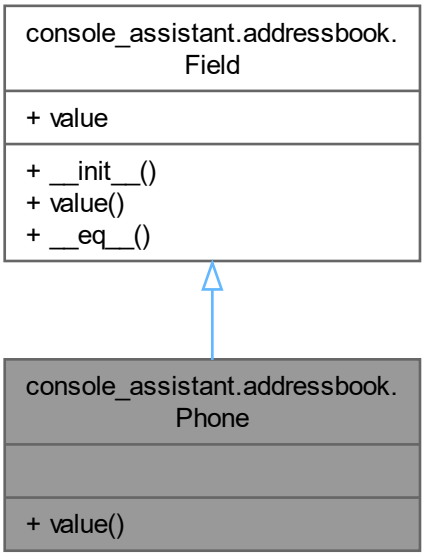
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/notebook.py

7.13 console_assistant.addressbook.Phone Class Reference

Inheritance diagram for console_assistant.addressbook.Phone:



Collaboration diagram for console_assistant.addressbook.Phone:



Public Member Functions

- def value (self, value)

Public Member Functions inherited from console_assistant.addressbook.Field

- def __init__ (self, str value)
- def value (self)
- def __eq__ (self, other)

Additional Inherited Members

Public Attributes inherited from console_assistant.addressbook.Field

- value

7.13.1 Detailed Description

```
--- '         (Record)
  .
```

Definition at line 25 of file addressbook.py.

7.13.2 Member Function Documentation

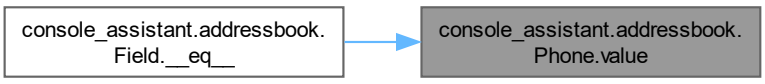
7.13.2.1 value()

```
def console_assistant.addressbook.Phone.value (
              self,
              value )
```

Reimplemented from console_assistant.addressbook.Field.

Definition at line 30 of file addressbook.py.

Here is the caller graph for this function:

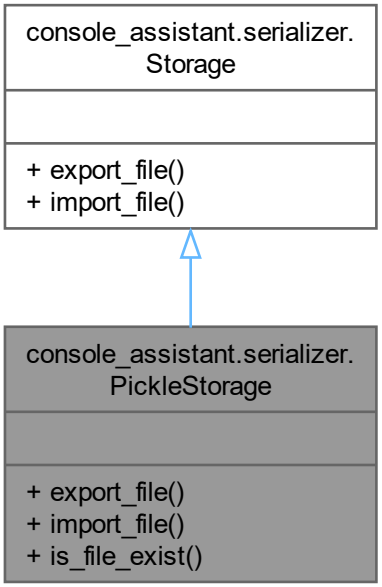


The documentation for this class was generated from the following file:

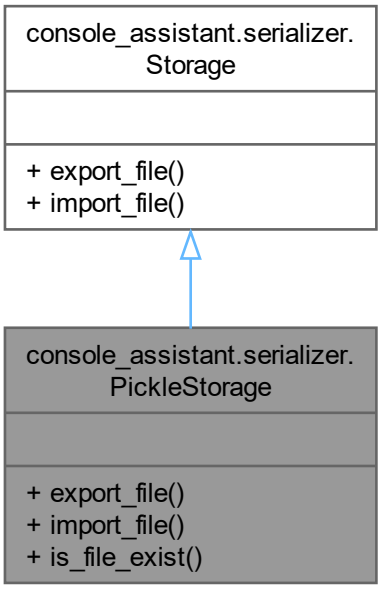
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/addressbook.py

7.14 console_assistant.serializer.PickleStorage Class Reference

Inheritance diagram for console_assistant.serializer.PickleStorage:



Collaboration diagram for console_assistant.serializer.PickleStorage:



Static Public Member Functions

- def export_file (obj, filename)
- def import_file (filename)
- def is_file_exist (filename)

Additional Inherited Members

- def export_file (object, str filename)
- def import_file (object, str filename)

7.14.1 Detailed Description

Definition at line 15 of file serializer.py.

7.14.2 Member Function Documentation

7.14.2.1 export_file()

```
def console_assistant.serializer.PickleStorage.export_file (
              obj,
              filename )  [static]
```

Reimplemented from console_assistant.serializer.Storage.

Definition at line 17 of file serializer.py.

7.14.2.2 import_file()

```
def console_assistant.serializer.PickleStorage.import_file (
              filename )  [static]
```

Reimplemented from console_assistant.serializer.Storage.

Definition at line 23 of file serializer.py.

7.14.2.3 is_file_exist()

```
def console_assistant.serializer.PickleStorage.is_file_exist (
              filename )  [static]
```

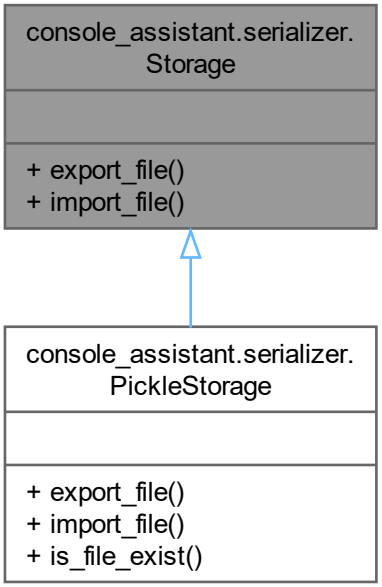
Definition at line 29 of file serializer.py.

The documentation for this class was generated from the following file:

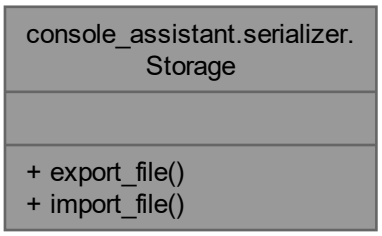
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/serializer.py

7.15 console_assistant.serializer.Storage Class Reference

Inheritance diagram for console_assistant.serializer.Storage:



Collaboration diagram for console_assistant.serializer.Storage:



Public Member Functions

- def export_file (object, str filename)
- def import_file (object, str filename)

7.15.1 Detailed Description

Definition at line 7 of file serializer.py.

7.15.2 Member Function Documentation

7.15.2.1 export_file()

```
def console_assistant.serializer.Storage.export_file (
              object,
             str filename )
```

Reimplemented in console_assistant.serializer.PickleStorage.

Definition at line 8 of file serializer.py.

7.15.2.2 import_file()

```
def console_assistant.serializer.Storage.import_file (
              object,
             str filename )
```

Reimplemented in console_assistant.serializer.PickleStorage.

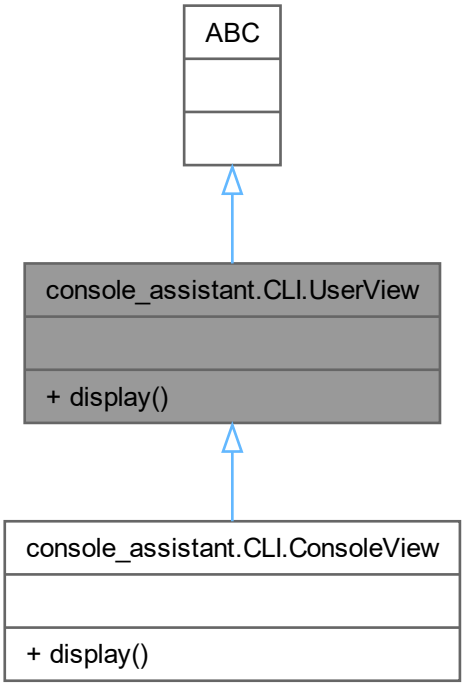
Definition at line 11 of file serializer.py.

The documentation for this class was generated from the following file:

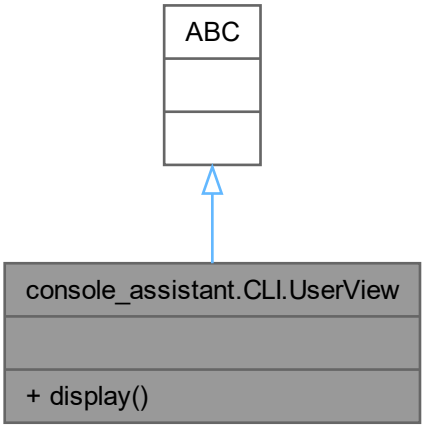
- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/serializer.py

7.16 console_assistant.CLI.UserView Class Reference

Inheritance diagram for console_assistant.CLI.UserView:



Collaboration diagram for console_assistant.CLI.UserView:



Public Member Functions

- def display (self, data)

7.16.1 Detailed Description

Definition at line 820 of file CLI.py.

7.16.2 Member Function Documentation

7.16.2.1 display()

```
def console_assistant.CLI.UserView.display (
              self,
              data )
```

Reimplemented in console_assistant.CLI.ConsoleView.

Definition at line 822 of file CLI.py.

The documentation for this class was generated from the following file:

- D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/CLI.py

Chapter 8

File Documentation

8.1 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/__init__.py File Reference

8.2 __init__.py

Go to the documentation of this file.

8.3 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/addressbook.py File Reference

Classes

- class console_assistant.addressbook.Field
- class console_assistant.addressbook.Phone
- class console_assistant.addressbook.Birthday
- class console_assistant.addressbook.Email
- class console_assistant.addressbook.AddressBook

Namespaces

- namespace console_assistant
- namespace console_assistant.addressbook

Variables

- namedtuple console_assistant.addressbook.Record = namedtuple("Record", ["name", "birthday", "phones", "emails", "address"])

8.4 addressbook.py

Go to the documentation of this file.

```
00001 import re
00002 from datetime import datetime, timedelta
00003 from collections import namedtuple, UserList
00004
00005
00006 class Field:
00007     """      ,    ,
00008        ."""
00009
00010     def __init__(self, value: str):
00011         self.__value = value
00012         self.valuevalue = value
00013
00014     @property
00015     def value(self):
00016         return self.__value
00017
00018     def __eq__(self, other):
00019         return self.valuevalue == other.value
00020
00021     # def __str__(self):
00022     #     return self.value
00023
00024
00025 class Phone(Field):
00026     """ --- '         (Record)
00027       ."""
00028
00029     @Field.value.setter
00030     def value(self, value):
00031         if not bool(re.match(r"\d{10}", value)) or len(value) > 10:
00032             raise ValueError("Phone number must be 10 digits")
00033         self.__value__value = value
00034
00035
00036 class Birthday(Field):
00037     """ --- '     ."""
00038
00039     @Field.value.setter
00040     def value(self, value):
00041         try:
00042             date = datetime.strptime(value, "%d.%m.%Y")
00043         except (TypeError, ValueError):
00044             raise ValueError("Invalid date format. Please use DD.MM.YYYY")
00045         if date > datetime.today():
00046             raise ValueError("Date cannot be in the future")
00047         self.__value__value = value
00048
00049
00050 class Email(Field):
00051     """ --- '    email"""
00052
00053     @Field.value.setter
00054     def value(self, value):
00055         if not bool(re.match(r"[\w.-]+@[a-zA-Z]+\.[a-zA-Z]{2,}", value)):
00056             raise ValueError("Email is not valid")
00057         self.__value__value = value
00058
00059
00060 Record = namedtuple("Record", ["name", "birthday", "phones", "emails", "address"])
00061
00062
00063 class AddressBook(UserList):
00064     def update(self, records):
00065         self.data.clear()
00066         self.data.extend(records)
00067
00068     def add_record(self, name, birthday=None, phones=None, emails=None, address=None):
00069         # ,
00070         for record in self.data:
00071             if record.name == name:
00072                 # # print(f"Contact with {name} already exist!")
00073                 # raise ValueError("Contact already exist in AddressBook.")
00074                 if birthday:
00075                     record.birthday.extend([Birthday(bd) for bd in birthday])
00076                 if phones:
00077                     record.phones.extend([Phone(ph) for ph in phones])
00078                 if emails:
00079                     record.emails.extend([Email(em) for em in emails])
00080                 if address:
00081                     record.address.extend(address)
00082                 return
```

```
00083
00084         #       ,
00085         birthday = [Birthday(birthday) for birthday in birthday] if birthday else []
00086         phones = [Phone(phone) for phone in phones] if phones else []
00087         emails = [Email(email) for email in emails] if emails else []
00088         address = [address] if address else []
00089         record = Record(
00090             name=name,
00091             birthday=birthday,
00092             phones=phones,
00093             emails=emails,
00094             address=address,
00095         )
00096         self.data.append(record)
00097
00098     def add_phone(self, name, phone):
00099         for record in self.data:
00100             if record.name == name:
00101                 record.phones.append(Phone(phone))
00102                 return
00103         self.add_record(name=name, phones=[phone])
00104
00105     def add_email(self, name, email):
00106         for record in self.data:
00107             if record.name == name:
00108                 record.emails.append(Email(email))
00109                 return
00110         self.add_record(name=name, emails=[email])
00111
00112     def add_address(self, name, address):
00113         for record in self.data:
00114             if record.name == name:
00115                 record.address.clear()
00116                 record.address.append(address)
00117                 return
00118         self.add_record(name=name, address=address)
00119
00120     def add_birthday(self, name, birthday):
00121         for record in self.data:
00122             if record.name == name:
00123                 record.birthday.clear()
00124                 record.birthday.append(Birthday(birthday))
00125                 return
00126         self.add_record(name=name, birthday=[birthday])
00127
00128     def remove_record(self, name):
00129         for record in self.data:
00130             if record.name == name:
00131                 self.data.remove(record)
00132                 return True
00133         return False
00134
00135     def update_name(self, name, new_name):
00136         for i, record in enumerate(self.data):
00137             if record.name == name:
00138                 self.data[i] = record._replace(name=new_name)
00139                 return True
00140         return False
00141
00142     def remove_address(self, name):
00143         for record in self.data:
00144             if record.name == name:
00145                 record.address.clear()
00146                 return True
00147         return False
00148
00149
00150     def delete_phone_by_index(self, name, phone_index):
00151         for record in self.data:
00152             if record.name == name:
00153                 del record.phones[phone_index]
00154                 return True
00155         return False
00156
00157     def delete_email_by_index(self, name, email_index):
00158         for record in self.data:
00159             if record.name == name:
00160                 del record.emails[email_index]
00161                 return True
00162         return False
00163
00164     def upcoming_birthdays(self, days):
00165         """         days """
00166         today = datetime.today().date()
00167         upcoming = today + timedelta(days=days)
00168         upcoming_bdays = AddressBook()
00169         for record in self.data:
```

```
00170             if record.birthday:
00171                 birthday = (
00172                     datetime.strptime(record.birthday[0].value, "%d.%m.%Y")
00173                     .replace(year=today.year)
00174                     .date()
00175                 )
00176                 if birthday.month == 2 and birthday.day == 29:
00177                     #
00178                     birthday = datetime(birthday.year + 1, 3, 1).date()
00179                 else:
00180                     birthday = birthday.replace(year=today.year)
00181                 if today <= birthday <= upcoming:
00182                     upcoming_bdays.data.append(record)
00183         return upcoming_bdays
00184 
00185     def find_records(self, search_term):
00186         found_contacts = AddressBook()
00187 
00188         for record in self.data:
00189             if (
00190                 search_term in record.name
00191                 or search_term in str(record.address)
00192                 or any(search_term in birthday.value for birthday in record.birthday)
00193                 or any(search_term in phone.value for phone in record.phones)
00194                 or any(search_term in email.value for email in record.emails)
00195             ):
00196                 found_contacts.append(record)
00197 
00198         return found_contacts
00199 
00200     def find_contact_by_name(self, name):
00201         for record in self.data:
00202             if record.name == name:
00203                 return [record]
00204         return None
00205 
00206     def iterator(self, n: int = 10):
00207         """          n-."""
00208 
00209         items = sorted(self.data)
00210         for i in range(0, len(items), n):
00211             data_slice = items[i : i + n]
00212             yield data_slice
00213             if i + n < len(items):
00214                 yield "continue"
00215 
```

8.5 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/CLI.py File Reference

Classes

- class console_assistant.CLI.Command
- class console_assistant.CLI.CommandCompleter
- class console_assistant.CLI.CommandParser
- class console_assistant.CLI.CommandExecutor
- class console_assistant.CLI.InputReader
- class console_assistant.CLI.UserView
- class console_assistant.CLI.ConsoleView

Namespaces

- namespace console_assistant
- namespace console_assistant.CLI

Functions

- def console_assistant.CLI.input_error (func)
- def console_assistant.CLI.hello (*args)
- def console_assistant.CLI.good_bye (*args)
- def console_assistant.CLI.save (*args)
- def console_assistant.CLI.load (*args)
- def console_assistant.CLI.parse_contact_params (string)
- def console_assistant.CLI.add_contact (*args)
- def console_assistant.CLI.remove_contact (*args)
- def console_assistant.CLI.remove_phone (*args)
- def console_assistant.CLI.remove_email (*args)
- def console_assistant.CLI.remove_address (*args)
- def console_assistant.CLI.upcoming_birthdays (*args)
- def console_assistant.CLI.change_name (*args)
- def console_assistant.CLI.search_contact (*args)
- def console_assistant.CLI.show_contact (*args)
- def console_assistant.CLI.build_contacts_table (contacts)
- def console_assistant.CLI._get_phones_str (phones)
- def console_assistant.CLI._get_emails_str (emails)
- def console_assistant.CLI.show_contacts (*args)
- def console_assistant.CLI.add_note (*args)
- def console_assistant.CLI.remove_note (*args)
- def console_assistant.CLI.add_tag (*args)
- def console_assistant.CLI.build_notes_table (notes, original_indices=False)
- def console_assistant.CLI.show_notes (*args)
- def console_assistant.CLI.search_notes (*args)
- def console_assistant.CLI.save_notes (*args)
- def console_assistant.CLI.change_note (*args)
- def console_assistant.CLI.remove_tag (*args)
- def console_assistant.CLI.load_notes (*args)
- def console_assistant.CLI.get_currency_table (CurrencyList currency_list)
- def console_assistant.CLI.get_currency (*args)
- def console_assistant.CLI.get_weather (*args)
- def console_assistant.CLI.help_commands (*args)
- def console_assistant.CLI.sort_folder_cli (*args)
- def console_assistant.CLI.cls (*args)
- def console_assistant.CLI.main ()

Variables

- dict console_assistant.CLI.COMMANDS
- AddressBook console_assistant.CLI.contacts = AddressBook()
- Notebook console_assistant.CLI.notebook = Notebook()
- str console_assistant.CLI.NOTES_FILE = "notes.bin"
- str console_assistant.CLI.CONTACT_FILE = "contacts.bin"
- f console_assistant.CLI.HELLO_MESSAGE = f"{N}Hello, I'm an assistant v1.0.0 {G}(c) Team-9, GoIT 2023.{N}\nType {Y}help{N} for more information.{N}"

8.6 CLI.py

Go to the documentation of this file.

```
00001 """    """
00002 from abc import ABC, abstractmethod
00003 import re
00004 import os
00005 import os.path
00006 from pathlib import Path
00007 from difflib import get_close_matches
00008
00009 from prompt_toolkit import PromptSession
00010 from prompt_toolkit.completion import Completion, Completer
00011 from prompt_toolkit.shortcuts import clear
00012
00013 from prettytable.colortable import ColorTable, Themes
00014
00015 from pygments import highlight
00016 from pygments.lexers import get_lexer_by_name
00017 from pygments.formatters import TerminalFormatter
00018 from prompt_toolkit.auto_suggest import AutoSuggestFromHistory
00019
00020 from bs4 import BeautifulSoup
00021 import requests
00022
00023 from console_assistant.addressbook import AddressBook
00024
00025 from console_assistant.notebook import Notebook
00026
00027 from console_assistant.curreny import CurrencyList
00028
00029 from console_assistant.serializer import PickleStorage
00030
00031 from console_assistant.filesorter import sort_folder
00032
00033
00034 from console_assistant.colors import *
00035
00036
00037 # ================================ Decorator ================================#
00038
00039
00040 def input_error(func):
00041     def wrapper(*func_args, **func_kwargs):
00042         try:
00043             return func(*func_args, **func_kwargs)
00044         except KeyError as error:
00045             return "{}".format(R + str(error).strip("'") + N)
00046         except ValueError as error:
00047             return f"{R+str(error)+N}"
00048         except TypeError as error:
00049             return f"{R + str(error) + N}"
00050         except FileNotFoundError:
00051             return R + "File not found" + N
00052         except IndexError:
00053             return R + "No such index" + N
00054
00055     return wrapper
00056
00057
00058 # ================================ handlers ================================#
00059
00060
00061 def hello(*args):
00062     return f"{G}How can I help you?{N}"
00063
00064
00065 def good_bye(*args):
00066     os.system("cls" if os.name == "nt" else "clear")
00067     save(CONTACT_FILE)
00068     save_notes(NOTES_FILE)
00069     print("See you later.\nDotn't warry, your data was saved.")
00070     return "Good bye!"
00071
00072
00073 @input_error
00074 def save(*args):
00075     if not args[0]:
00076         raise ValueError("Give me a filename.")
00077
00078     home_path = Path.home()
00079     file_path = home_path / args[0]
00080     PickleStorage.export_file(contacts, file_path)
00081     return f"File {args[0]} saved."
00082
```

```
00083
00084 @input_error
00085 def load(*args):
00086     if not args[0]:
00087         raise ValueError("Give me a filename.")
00088
00089     home_path = Path.home()
00090     file_path = home_path / args[0]
00091     if PickleStorage.is_file_exist(file_path):
00092         contacts.clear()
00093         contacts.update(PickleStorage.import_file(file_path))
00094         return f"File {args[0]} loaded."
00095     else:
00096         raise FileNotFoundError
00097
00098
00099 # ========================    ============================ #
00100
00101
00102 def parse_contact_params(string):
00103     phone_regex = re.compile(r"\d{10}")
00104     date_regex = re.compile(r"\d{2}\.\d{2}\.\d{4}")
00105     email_regex = re.compile(r"[\w.-]+@[a-zA-Z]+\.[a-zA-Z]{2,}")
00106
00107     phone = phone_regex.search(string)
00108     date = date_regex.search(string)
00109     email = email_regex.search(string)
00110
00111     phone = phone.group(0) if phone else None
00112     date = date.group(0) if date else None
00113     email = email.group(0) if email else None
00114
00115     return phone, date, email
00116
00117
00118 @input_error
00119 def add_contact(*args):
00120     """    """
00121     usage_message = (
00122         f"Example of usage: {G}add contact {Y}Username{N} [phone1] [birthday] [email]."
00123     )
00124     error_message = None
00125     name = args[0]
00126     if not name:
00127         error_message = "Please provide a name for the contact."
00128
00129     if error_message:
00130         print(usage_message)
00131         raise ValueError(error_message)
00132
00133     contacts.add_record(name)
00134     rest_args = " ".join(args[1:]) if args[1:] != (None,) else False
00135
00136     if rest_args:
00137         parsed_params = parse_contact_params(" ".join(args[1:]))
00138         phone = parsed_params[0]
00139         birthday = parsed_params[1]
00140         email = parsed_params[2]
00141
00142         if phone:
00143             contacts.add_phone(name, phone)
00144
00145         if birthday:
00146             contacts.add_birthday(name, birthday)
00147
00148         if email:
00149             contacts.add_email(name, email)
00150
00151     return f"I added a info to contact {name} to the address book."
00152
00153
00154 @input_error
00155 def remove_contact(*args):
00156     """-handler    ."""
00157
00158     usage_message = f"Example of usage: {G}remove contact {Y}Username{N}."
00159     error_message = None
00160
00161     if not args[0]:
00162         error_message = "Give me a name, please."
00163
00164     if error_message:
00165         print(usage_message)
00166         raise KeyError(error_message)
00167
00168     result = contacts.remove_record(args[0])
00169
```

```
00170     if result:
00171         return f"Contact {args[0]} was removed."
00172     return f"{R}Contact {args[0]} not in address book{N}."
00173 
00174 
00175 # @input_error
00176 # def set_phone(*args):
00177 #     """       ."""
00178 
00179 #     usage_message = f"Example of usage: {G}set phone {Y}Username 0985467856{N}"
00180 #     error_messageK, error_messageV = None, None
00181 
00182 #     if not args[0]:
00183 #         error_messageK = "Give me a name, please."
00184 #     if not args[1]:
00185 #         error_messageV = "Give me a phone, please."
00186 
00187 #     if error_messageK:
00188 #         print(usage_message)
00189 #         raise KeyError(error_messageK)
00190 #     if error_messageV:
00191 #         print(usage_message)
00192 #         raise ValueError(error_messageV)
00193 
00194 #     contacts.add_phone(args[0], args[1])
00195 
00196 #     return f"I added a phone {args[1]} to contact {args[0]}"
00197 
00198 
00199 @input_error
00200 def remove_phone(*args):
00201     """       ."""
00202 
00203     usage_message = f"Example of usage: {G}remove phone {Y}Username 1{N}"
00204     error_message = None
00205     if not args[0]:
00206         error_message = "Give me a name, please."
00207     elif not args[1]:
00208         error_message = "Give me an index of phone, please."
00209     elif not args[1].isdigit():
00210         error_message = "Index of phone must be a number."
00211     if error_message:
00212         print(usage_message)
00213         raise ValueError(error_message)
00214 
00215     result = contacts.delete_phone_by_index(args[0], int(args[1]) - 1)
00216     if result:
00217         return f"I removed a phone of contact {args[0]}"
00218     return f"{R}No contact {args[0]} in AddressBook.{N}"
00219 
00220 
00221 # @input_error
00222 # def set_email(*args):
00223 #     """ email     ."""
00224 
00225 #     usage_message = f"Example of usage: {G}set email {Y}Username my_mail@i.ua{N}"
00226 #     error_messageK, error_messageV = None, None
00227 
00228 #     if not args[0]:
00229 #         error_messageK = "Give me a name, please"
00230 
00231 #     if not args[1]:
00232 #         error_messageV = "Give me a email, please"
00233 
00234 #     if error_messageK:
00235 #         print(usage_message)
00236 #         raise KeyError(error_messageK)
00237 #     if error_messageV:
00238 #         print(usage_message)
00239 #         raise ValueError(error_messageV)
00240 
00241 #     contacts.add_email(args[0], args[1])
00242 
00243 #     return f"I added a email {args[1]} to contact {args[0]}"
00244 
00245 
00246 @input_error
00247 def remove_email(*args):
00248     """ email     ."""
00249 
00250     usage_message = f"Example of usage: {G}remove email {Y}Username 1{N}"
00251     error_message = None
00252     if not args[0]:
00253         error_message = "Give me a name, please"
00254     elif not args[1]:
00255         error_message = "Give me an index of email, please."
00256     elif not args[1].isdigit():
```

```
00257         error_message = "Index of email must be a number."
00258     if error_message:
00259         print(usage_message)
00260         raise ValueError(error_message)
00261 
00262     result = contacts.delete_email_by_index(args[0], int(args[1]) - 1)
00263 
00264     if result:
00265         return f"I removed a email of contact {args[0]}"
00266     return f"{R}No contact {args[0]} in AddressBook.{N}"
00267 
00268 
00269 # @input_error
00270 # def set_address(*args):
00271 #     """      ."""
00272 
00273 #     usage_message = f"Example of usage: {G}set address {Y}Username Address of user{N}"
00274 #     error_messageK, error_messageV = None, None
00275 
00276 #     if not args[0]:
00277 #         error_messageK = "Give me a name, please"
00278 
00279 #     if not args[1]:
00280 #         error_messageV = "Give me an address, please"
00281 
00282 #     if error_messageK:
00283 #         print(usage_message)
00284 #         raise KeyError(error_messageK)
00285 #     if error_messageV:
00286 #         print(usage_message)
00287 #         raise ValueError(error_messageV)
00288 
00289 #     contacts.add_address(args[0], args[1])
00290 
00291 #     return f"I added a address {args[1]} to contact {args[0]}"
00292 
00293 
00294 @input_error
00295 def remove_address(*args):
00296     """ email    ."""
00297 
00298     usage_message = f"Example of usage: {G}set address {Y}Username Address of user{N}"
00299     error_message = None
00300 
00301     if not args[0]:
00302         error_message = "Give me a name, please."
00303 
00304     if error_message:
00305         print(usage_message)
00306         raise ValueError(error_message)
00307 
00308     result = contacts.remove_address(args[0])
00309     if result:
00310         return f"I removed a address of contact {args[0]}"
00311     return f"{R}No contact {args[0]} in AddressBook.{N}"
00312 
00313 
00314 # @input_error
00315 # def set_birthday(*args):
00316 #     """-handler     ."""
00317 
00318 #     usage_message = f"Example of usage: {G}set birthday {Y}Username 13.03.1989{N}"
00319 #     error_messageK, error_messageV = None, None
00320 
00321 #     if not args[0]:
00322 #         error_messageK = "Give me a name, please"
00323 
00324 #     if not args[1]:
00325 #         error_messageV = "Give me an birthday in format DD.MM.YYYY, please"
00326 
00327 #     if error_messageK:
00328 #         print(usage_message)
00329 #         raise KeyError(error_messageK)
00330 #     if error_messageV:
00331 #         print(usage_message)
00332 #         raise ValueError(error_messageV)
00333 
00334 #     contacts.add_birthday(args[0], args[1])
00335 
00336 #     return f"I added a birthday {args[1]} to contact {args[0]}"
00337 
00338 
00339 @input_error
00340 def upcoming_birthdays(*args):
00341     if not args[0]:
00342         raise TypeError("Set days you interested")
00343     days = int(args[0])
```

```
00344     result = contacts.upcoming_birthdays(days)
00345     return f"{N}{build_contacts_table(result)}{N}"
00346
00347
00348 @input_error
00349 def change_name(*args):
00350     usage_message = f"Example of usage: {G}change name {Y}Old_name New_name{N}"
00351     error_message = None
00352
00353     if not args[0]:
00354         error_message = "Give me a some name, please"
00355
00356     if not args[1]:
00357         error_message = "Give me a new name, please"
00358
00359     if error_message:
00360         print(usage_message)
00361         raise ValueError(error_message)
00362
00363     result = contacts.update_name(args[0], args[1])
00364     if result:
00365         return f"I updatw name {args[0]} -> {args[1]}"
00366     return f"{R}No contact {args[0]} in AddressBook.{N}"
00367
00368
00369 @input_error
00370 def search_contact(*args):
00371     if not args[0]:
00372         raise KeyError("Give me a some name, please")
00373
00374     results = contacts.find_records(args[0])
00375
00376     if results:
00377         return f"{N}{build_contacts_table(results)}{N}"
00378     return "By your request found nothing"
00379
00380
00381 @input_error
00382 def show_contact(*args):
00383     if not args[0]:
00384         raise KeyError("Give me a some name, please")
00385     result = contacts.find_contact_by_name(args[0])
00386     if result is not None:
00387         return f"{N}{build_contacts_table(result)}{N}"
00388     return f"{R}Contact {args[0]} not found.{N}"
00389
00390
00391 def build_contacts_table(contacts):
00392     table = ColorTable(theme=Themes.OCEAN)
00393     table.field_names = ["#", "Name", "Birthday", "Phones", "Emails", "Address"]
00394     table.align["Emails"] = "l"
00395     # table.set_style(SINGLE_BORDER)
00396     for i, record in enumerate(contacts):
00397         birthday = record.birthday[0].value if record.birthday else "-"
00398         address = record.address[0] if record.address else "-"
00399         phones_str = _get_phones_str(record.phones)
00400         emails_str = _get_emails_str(record.emails)
00401         table.add_row(
00402             [
00403                 f"{W}{i + 1}{N}",
00404                 f"{G}{record.name}{N}",
00405                 f"{B}{birthday}{N}",
00406                 phones_str,
00407                 emails_str,
00408                 f"{Y}{address}{N}",
00409             ],
00410             divider=True,
00411         )
00412     return table
00413
00414
00415 def _get_phones_str(phones):
00416     if not phones:
00417         return "-"
00418     phones_str = ""
00419     for i, phone in enumerate(phones):
00420         phones_str += f"{W}{i+1}. {B}{phone.value}{N}\n"
00421     return phones_str[:-1]
00422
00423
00424 def _get_emails_str(emails):
00425     if not emails:
00426         return "-"
00427     emails_str = ""
00428     for i, email in enumerate(emails):
00429         emails_str += f"{W}{i+1}. {P}{email.value}{N}\n"
00430     return emails_str[:-1]
```

```
00431
00432
00433 @input_error
00434 def show_contacts(*args):
00435     number_of_entries = (
00436         int(args[0])
00437         if args[0] is not None and isinstance(args[0], str) and args[0].isdigit()
00438         else 20
00439     )
00440
00441     current_contact_num = 1  #
00442     for tab in contacts.iterator(number_of_entries):
00443         if tab == "continue":
00444             input(G + "Press <Enter> to continue..." + N)
00445         else:
00446             table = build_contacts_table(tab)
00447             # table.align["Emails"] = "l"
00448             #      #
00449             for i, row in enumerate(table._rows):
00450                 row[0] = current_contact_num + i
00451             print(table)
00452             #
00453             current_contact_num += len(tab)
00454     return f"Address book contain {len(contacts)} contact(s)."
00455
00456
00457 # ============================    ======================== #
00458
00459
00460 @input_error
00461 def add_note(*args):
00462     usage_message = f"Example of usage: {G}add note {Y}Tag Text{N}"
00463     error_message = None
00464     if args[0] is None:
00465         error_message = "Give me a tag and text, please."
00466     elif args[0] is not None and args[0].isdigit():
00467         error_message = "Tag cannot be a number."
00468     elif not args[1]:
00469         error_message = "Give me a text, please."
00470     if error_message:
00471         print(usage_message)
00472         raise ValueError(error_message)
00473
00474     notebook.add_note([args[0]], args[1])
00475     return "I added note"
00476
00477
00478 @input_error
00479 def remove_note(*args):
00480     if not args[0]:
00481         raise ValueError("Give me an index first.")
00482     if not args[0].isdigit():
00483         raise ValueError("Index must be a number.")
00484     notebook.remove_note(int(args[0]))
00485     return "I removed note"
00486
00487
00488 @input_error
00489 def add_tag(*args):
00490     usage_message = f"Example of usage: {G}add tag {Y}1 Tag{N}"
00491     error_message = None
00492     if args[0] is None:
00493         error_message = "Give me an index first, please."
00494     elif not args[0].isdigit():
00495         error_message = "Index must be a number."
00496     elif args[1] is None:
00497         error_message = "Give me a tag, please."
00498     elif args[1].isdigit():
00499         error_message = "Tag cannot be a number."
00500
00501     if error_message:
00502         print(usage_message)
00503         raise ValueError(error_message)
00504
00505     notebook.add_tag(int(args[0]), args[1])
00506     return f"I added tag {args[1]} to note {args[0]}."
00507
00508
00509 def build_notes_table(notes, original_indices=False):
00510     table = ColorTable(theme=Themes.OCEAN)
00511     table.field_names = ["Index", "Tags", "Creation Date", "Text"]
00512     table.max_width["Text"] = 79
00513     # table.set_style(SINGLE_BORDER)
00514     for note, index in notes:
00515         if original_indices:
00516             index = notebook.data.index(note)
00517         date_str = note.date.strftime("%Y-%m-%d %H:%M:%S")
```

```
00518         table.add_row(
00519             [
00520                 f"{W}{index}{N}",
00521                 G + ", ".join(note.tags) + N,
00522                 f"{Y}{date_str}{N}",
00523                 f"{B}{note.text}{N}",
00524             ],
00525             divider=True,
00526         )
00527     return table
00528
00529
00530 # @input_error
00531 def show_notes(*args):
00532     if len(notebook) != 0:
00533         if args[0] is None or not args[0].isdigit():
00534             notes = notebook.display_notes(tag=args[0] or None, original_indices=True)
00535             print(build_notes_table(notes, original_indices=True))
00536         else:
00537             n = int(args[0])
00538             for i, tab in enumerate(notebook.iterator_notes(n)):
00539                 if tab == "continue":
00540                     input(G + "Press <Enter> to continue..." + N)
00541                 else:
00542                     table = build_notes_table(tab)
00543                     print(table)
00544     return f"Notes book contain {len(notebook)} note(s)."
00545
00546
00547 @input_error
00548 def search_notes(*args):
00549     if not args[0]:
00550         raise KeyError("Please, add searh query")
00551     results = notebook.find_notes(args[0])
00552     if not results:
00553         return f"{R}Nothing found for {args[0]}{N}"
00554     return build_notes_table(results)
00555
00556
00557 @input_error
00558 def save_notes(*args):
00559     if not args[0]:
00560         raise ValueError("Give me a filename")
00561     home_path = Path.home()
00562     file_path = home_path / args[0]
00563     PickleStorage.export_file(notebook, file_path)
00564     return f"File {args[0]} saved"
00565
00566
00567 @input_error
00568 def change_note(*args):
00569     if not args[0]:
00570         raise KeyError("Please, set integer index")
00571
00572     notebook.change_note(int(args[0]), args[1])
00573
00574     return f"I changed note {args[0]}"
00575
00576
00577 @input_error
00578 def remove_tag(*args):
00579     if not args[0]:
00580         raise ValueError("Give me an index first.")
00581     if not args[0].isdigit():
00582         raise ValueError("Index must be a number.")
00583     if not args[1]:
00584         raise ValueError("Give me a tag, please.")
00585
00586     result = notebook.remove_tag(int(args[0]), args[1])
00587     if result:
00588         return f"I changed tag {args[1]} for note with index {args[0]}"
00589     else:
00590         raise ValueError(f"Tag {args[1]} not found for this note")
00591
00592
00593 @input_error
00594 def load_notes(*args):
00595     if not args[0]:
00596         raise ValueError("Give me a filename")
00597
00598     home_path = Path.home()
00599     file_path = home_path / args[0]
00600     if PickleStorage.is_file_exist(file_path):
00601         notebook.clear()
00602         notebook.update(PickleStorage.import_file(file_path))
00603         return f"File {args[0]} loaded"
00604     else:
```

```
00605         raise FileNotFoundError
00606
00607
00608 # ================================  ================================== #
00609
00610
00611 def get_currency_table(currency_list: CurrencyList):
00612     table = ColorTable(theme=Themes.OCEAN)
00613     table.max_width["Currency"] = 30
00614     table.max_width["Short Name"] = 15
00615     table.max_width["Rate"] = 10
00616     table.align["Short Name"] = "c"
00617     table.align["Rate"] = "c"
00618     table.field_names = ["Currency", "Short Name", "Rate"]
00619     for currency in currency_list.get_currency_rates():
00620         table.add_row([currency.name, currency.cc, currency.rate])
00621     return table
00622
00623
00624 @input_error
00625 def get_currency(*args):
00626     return get_currency_table(CurrencyList())
00627
00628
00629 # ================================  ================================== #
00630
00631 #!
00632
00633
00634 def get_weather(*args):
00635     city = args[0]
00636     url = f"https://ua.sinoptik.ua/-{city}"
00637     response = requests.get(url)
00638     soup = BeautifulSoup(response.content, "html.parser")
00639     temperature = soup.select_one(".today-temp")
00640     description = soup.select_one(".description")
00641
00642     if temperature and description:
00643         temperature = temperature.text
00644         description = description.text.strip()
00645
00646         table = ColorTable(theme=Themes.OCEAN)
00647         table.field_names = ["City", "Temperature", "Description"]
00648         table.add_row([f"{Y}{city}{N}", f"{G}{temperature}{N}", description])
00649         table._max_width = {"Description": 79}
00650
00651         return table
00652     else:
00653         return f"Unable to find weather information for {city}."
00654
00655
00656 # ========================================================================== #
00657 def help_commands(*args):
00658     """    ."""
00659
00660     PACKAGE_ROOT = os.path.abspath(os.path.dirname(__file__))
00661     README_PATH = os.path.join(PACKAGE_ROOT, "../README.md")
00662
00663     if not os.path.exists(README_PATH):
00664         return R + f"File {README_PATH} not found." + N
00665
00666     with open(README_PATH, "r") as file:
00667         code = file.read()
00668         lexer = get_lexer_by_name("markdown")
00669         formatted_code = highlight(code, lexer, TerminalFormatter())
00670         return formatted_code
00671
00672
00673 @input_error
00674 def sort_folder_cli(*args):
00675     if not args[0]:
00676         raise ValueError("Give me a folder name, please.")
00677     return sort_folder(args[0])
00678
00679
00680 def cls(*args):
00681     clear()
00682     return HELLO_MESSAGE
00683
00684
00685 # ================================ handler loader ============================#
00686
00687 class Command:
00688     def __init__(self, name, handler, description=None, example=None):
00689         self.name = name
00690         self.description = description
00691         self.example = example
```

```
00692         self.handler = handler
00693
00694
00695 COMMANDS = {
00696     # --- Hello commands ---
00697     "help": Command("help", help_commands),
00698     "hello": Command("hello", hello),
00699     # --- Manage contacts ---
00700     "add contact": Command("add contact", add_contact),
00701     # "set phone": Command("set phone", set_phone),
00702     "remove phone": Command("remove phone", remove_phone),
00703     # "set email": Command("set email", set_email),
00704     "remove email": Command("remove email", remove_email),
00705     # "set address": Command("set address", set_address),
00706     "remove address": Command("remove address", remove_address),
00707     # "set birthday": Command("set birthday", set_birthday),
00708     "upcoming birthdays": Command("upcoming birthdays", upcoming_birthdays),
00709     "show contacts": Command("show contacts", show_contacts),
00710     "search contact": Command("search contact", search_contact),
00711     "show contact": Command("show contact", show_contact),
00712     "remove contact": Command("remove contact", remove_contact),
00713     "change name": Command("change name", change_name),
00714     "save": Command("save", save),
00715     "load": Command("load", load),
00716     # ---  ---
00717     "currency": Command("currency", get_currency),
00718     # ---  ---
00719     "weather in": Command("weather in", get_weather),
00720     # --- Manage notes ---
00721     "add note": Command("add note", add_note),
00722     "add tag": Command("add tag", add_tag),
00723     "remove note": Command("remove note", remove_note),
00724     "show notes": Command("show notes", show_notes),
00725     "save notes": Command("save notes", save_notes),
00726     "load notes": Command("load notes", load_notes),
00727     "search notes": Command("search notes", search_notes),
00728     "change note": Command("change note", change_note),
00729     "remove tag": Command("remove tag", remove_tag),
00730     # --- Sorting folder commnad ---
00731     "sort folder": Command("sort folder", sort_folder_cli),
00732     # --- Googd bye commnad ---
00733     "good bye": Command("good bye", good_bye),
00734     "close": Command("close", good_bye),
00735     "exit": Command("exit", good_bye),
00736     "cls": Command("cls", cls),
00737 }
00738
00739 class CommandCompleter(Completer):
00740     def get_completions(self, document, complete_event):
00741         text_before_cursor = document.current_line_before_cursor
00742         command, _, rest = text_before_cursor.partition(" ")
00743         if not rest:
00744             matches = [c for c in COMMANDS if c.startswith(command)]
00745             for m in matches:
00746                 yield Completion(m, display=m, start_position=-len(command))
00747
00748 class CommandParser:
00749     def __init__(self, commands):
00750         self.commands = commands
00751         self.command_pattern = "|".join(commands)
00752         self.pattern = re.compile(
00753             r"\b(\.|"
00754             + self.command_pattern
00755             + r")\b(?:\s+([--a-zA-Z0-9\.\:\\_\-]+))?(?:\s+(.+))?",
00756             re.IGNORECASE,
00757         )
00758
00759     def parse_command(self, command):
00760         text = self.pattern.search(command)
00761
00762         params = (
00763             tuple(
00764                 map(
00765                     lambda x: x.lower() if text.groups().index(x) == 0 else x,
00766                     text.groups(),
00767                 )
00768             )
00769             if text
00770             else (None, command, 0)
00771         )
00772
00773         return params
00774
00775
00776 class CommandExecutor:
00777     def __init__(self, commands):
00778         self.commands = commands
```

```
00779
00780     def execute_command(self, command, *args):
00781         handler = self.commands.get(command)
00782         if handler is None:
00783             if command not in COMMANDS:
00784                 matches = get_close_matches(args[0], COMMANDS)
00785                 if matches:
00786                     suggestion = matches[0]
00787                     return f"{W}Command {R + args[0] + N} {W}not found. Possibly you mean {Y +
      suggestion + N}?" # noqa
00788                 else:
00789                     return R + "What do you mean?" + N
00790             else:
00791                 return "Command not implemented"
00792         else:
00793             return handler.handler(*args)
00794
00795
00796 class InputReader:
00797     def __init__(self):
00798         self.session = PromptSession(completer=CommandCompleter(), complete_while_typing=True,
      auto_suggest=AutoSuggestFromHistory()) # noqa
00799
00800     def wait_for_input(self):
00801         while True:
00802             inp = self.session.prompt("»> ")
00803             if inp == "":
00804                 continue
00805             break
00806         return inp
00807
00808
00809
00810 # ================================ main function ============================ #
00811
00812 contacts = AddressBook()  # Global variable for storing contacts
00813 notebook = Notebook()  # Global variable for storing notes
00814
00815
00816 NOTES_FILE = "notes.bin"
00817 CONTACT_FILE = "contacts.bin"
00818 HELLO_MESSAGE = f"{N}Hello, I'm an assistant v1.0.0 {G}(c) Team-9, GoIT 2023.{N}\nType {Y}help{N} for
      more information.{N}" # noqa
00819
00820 class UserView(ABC):
00821     @abstractmethod
00822     def display(self, data):
00823         pass
00824 class ConsoleView(UserView):
00825     def display(self, data):
00826         print(data)
00827
00828
00829 def main():
00830     os.system("cls" if os.name == "nt" else "clear")
00831     print(HELLO_MESSAGE)
00832     load(CONTACT_FILE)
00833     load_notes(NOTES_FILE)
00834     parser = CommandParser(COMMANDS)
00835     executor = CommandExecutor(COMMANDS)
00836     reader = InputReader()
00837     console_view = ConsoleView()
00838     while True:
00839         command = reader.wait_for_input()
00840         if command.strip() == ".":
00841             save(CONTACT_FILE)
00842             save_notes(NOTES_FILE)
00843             return
00844         if command.strip() == "*":
00845             return
00846
00847         params = parser.parse_command(command)
00848         response = executor.execute_command(params[0], *params[1:])
00849         console_view.display(f"{G}{response}{N}")
00850
00851         if response == "Good bye!":
00852             return
00853
00854
00855 # ================================ main program ============================ #
00856
00857 if __name__ == "__main__":
00858     main()
```

8.7 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/colors.py File Reference

Namespaces

- namespace console_assistant
- namespace console_assistant.colors

Variables

- str console_assistant.colors.G = "\033[1;92m"
- str console_assistant.colors.B = "\033[1;96m"
- str console_assistant.colors.P = "\033[4;95m"
- str console_assistant.colors.R = "\033[1;91m"
- str console_assistant.colors.N = "\033[0m"
- str console_assistant.colors.Y = "\033[0;93m"
- str console_assistant.colors.W = "\033[97m"

8.8 colors.py

Go to the documentation of this file.

```
00001 G = "\033[1;92m"  # GREEN
00002 B = "\033[1;96m"  # Blue
00003 P = "\033[4;95m"  # Pink
00004 R = "\033[1;91m"  # Red
00005 N = "\033[0m"  # Reset
00006 Y = "\033[0;93m"  # Yellow
00007 W = "\033[97m"  # White
```

8.9 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/curreny.py File Reference

Classes

- class console_assistant.curreny.CurrencyList

Namespaces

- namespace console_assistant
- namespace console_assistant.curreny

Functions

- def console_assistant.curreny.get_currency_table (CurrencyList currency_list)

Variables

- namedtuple console_assistant.curreny.Currency = namedtuple("Currency", ["name", "rate", "cc"])
- CurrencyList console_assistant.curreny.cur = CurrencyList()

8.10 curreny.py

Go to the documentation of this file.

```
00001 from collections import namedtuple, UserList
00002 import requests
00003 from prettytable import PrettyTable
00004
00005
00006 Currency = namedtuple("Currency", ["name", "rate", "cc"])
00007
00008
00009 class CurrencyList(UserList):
00010     URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchangenew?json"
00011
00012     def __init__(self):
00013         super().__init__()
00014         self.refresh()
00015
00016     def refresh(self):
00017         response = requests.get(self.URL)
00018         data = response.json()
00019         for item in data:
00020             if item.get("cc") in ("USD", "EUR", "XAU", "XAG", "XPT", "XPD"):
00021                 currency = Currency(item.get("txt"), item.get("rate"), item.get("cc"))
00022                 self.data.append(currency)
00023
00024     def get_currency_by_cc(self, cc: str) -> Currency:
00025         for currency in self.data:
00026             if currency.cc == cc:
00027                 return currency
00028
00029     def get_currency_rates(self):
00030         return self.data
00031
00032
00033 if __name__ == "__main__":
00034
00035     def get_currency_table(currency_list: CurrencyList):
00036         table = PrettyTable()
00037         table.max_width["Currency"] = 30
00038         table.max_width["Short Name"] = 15
00039         table.max_width["Rate"] = 10
00040         table.align["Short Name"] = "c"
00041         table.align["Rate"] = "c"
00042         table.field_names = ["Currency", "Short Name", "Rate"]
00043         for currency in currency_list.get_currency_rates():
00044             table.add_row([currency.name, currency.cc, currency.rate])
00045         return table
00046
00047     cur = CurrencyList()
00048     get_currency_table(cur)
```

8.11 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/file_copies_deleter.py File Reference

Namespaces

- namespace console_assistant
- namespace console_assistant.file_copies_deleter

Functions

- def console_assistant.file_copies_deleter.count_files (abs_path, file_format="")
- def console_assistant.file_copies_deleter.get_hash (path_to_file)
- def console_assistant.file_copies_deleter.count_copies (dir_path, file_format="")
- def console_assistant.file_copies_deleter.delete_files (hash_dictionary)
- def console_assistant.file_copies_deleter.delete_empty_folders (path)
- def console_assistant.file_copies_deleter.copies_deleter (root)

8.12 file_copies_deleter.py

Go to the documentation of this file.

```
00001 import os
00002 from os import walk
00003 import hashlib
00004 import time
00005 from os.path import join, getsize
00006
00007 from .colors import G, R, N, Y
00008
00009 # Getting an absolut path to directory with files
00010 # def get_path_dir():
00011 #     abs_path = input("Please, enter the absolute path to directory: ")
00012 #     if len(abs_path) == 0:
00013 #         print("Directory is not specified. Try again.\n")
00014 #     elif os.path.isdir(abs_path):
00015 #         return abs_path
00016 #     else:
00017 #         print("Entered path doesn't exist. Try again.\n")
00018
00019
00020 def count_files(abs_path, file_format=""):
00021     total_count_files = 0
00022
00023     for root, _, files in walk(abs_path, topdown=True):
00024         for file in files:
00025             if file.endswith(file_format):
00026                 total_count_files += 1
00027
00028     print(f"{G}Files found:{N} {Y}{total_count_files}{N}")
00029
00030
00031 # The function takes absolute path to file and returns its hash
00032 def get_hash(path_to_file):
00033     blocksize = 65536
00034     file_hash = hashlib.md5(open(path_to_file, "rb").read(blocksize)).hexdigest()
00035     return file_hash
00036
00037
00038 # This function returns the number of copies found and the dictionary as
00039 # {hash_1: [path_to_file1, path_to_file1...], hash_2: [path_to_file3, path_to_file4...]...}
00040 def count_copies(dir_path, file_format=""):
00041     copy_count = 0
00042     hash_dict = {}
00043
00044     for root, _, files in walk(dir_path, topdown=True):
00045         for file in files:
00046             if file.endswith(file_format):
00047                 file_path = join(root, file)  # Absolut path to file
00048
00049                 file_hash = get_hash(file_path)  # File's hash
00050
00051                 if file_hash in hash_dict.keys():
00052                     copy_count += 1
00053                     hash_dict[file_hash].append(file_path)
00054                 else:
00055                     hash_dict[file_hash] = [file_path]
00056
00057     return copy_count, hash_dict
00058
00059
00060 # Check if the hash key is already in the dictionary; if it is, delete the new file by its path
00061 def delete_files(hash_dictionary):
00062     total_deleted = 0
00063     for path_dict in hash_dictionary.values():
00064         for file_path in path_dict[1:]:
00065             total_deleted += getsize(file_path) / (1024 * 1024)  # size in MB
00066             os.remove(file_path)  # Deleting a file
00067
00068     return round(total_deleted, 2)
00069
00070
00071 # The function deletes empty folders that may have formed after deleting copies
00072 def delete_empty_folders(path):
00073     for element in os.scandir(path):
00074         if os.path.isdir(element):
00075             full_path = os.path.join(path, element)
00076             delete_empty_folders(full_path)  # recursion
00077             if not os.listdir(full_path):
00078                 os.rmdir(full_path)  # remove empty folder
00079
00080
00081 def copies_deleter(root):
00082     path_to_dir = root
```

```
00083     count_files(path_to_dir)
00084
00085     print("Looking for copies...")
00086     time.sleep(0.5)
00087
00088     copy_count, hash_dict = count_copies(path_to_dir)
00089     print(f"{G}Copies found:{Y} {copy_count}{N}")
00090
00091     if copy_count != 0:
00092         while True:
00093             want_delete = input(
00094                 f"{G}Do you want to delete the detected copies (y/n)?{N} "
00095             )
00096
00097             if want_delete in ("yes", "YES", "Y", "y"):
00098                 total_deleted = delete_files(hash_dict)
00099                 print(f"Deleted {total_deleted} ")
00100                 time.sleep(3)
00101                 break
00102             elif want_delete in ("n", "N", "no", "NO"):
00103                 print(f"{G}Ok{N}")
00104                 time.sleep(2)
00105                 break
00106             else:
00107                 print(f"{R}You entered a non-existent command. Please try again.{N}\n")
00108
00109         delete_empty_folders(path_to_dir)
00110     return "Done"
00111
00112
00113 if __name__ == "__main__":
00114     copies_deleter("d:\\Different\\Garbage\\")
```

8.13 D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/filesorter.py File Reference

Namespaces

- namespace console_assistant
- namespace console_assistant.filesorter

Functions

- def console_assistant.filesorter.get_file_cathegory (str file)
- def console_assistant.filesorter.normalise_file_name (str file)
- def console_assistant.filesorter.create_folders (root)
- def console_assistant.filesorter.organize_files (path, level=0, known_exts=set(), unknown_exts=set(), categories=set())
- def console_assistant.filesorter.unpack (archive_path, path_to_unpack)
- def console_assistant.filesorter.remove_empty (path)
- def console_assistant.filesorter.known_exts (root)
- def console_assistant.filesorter.sort_folder (root)

Variables

- dict console_assistant.filesorter.EXT_FOLDER

8.14 filesorter.py

Go to the documentation of this file.

```
00001 import shutil
00002 from pathlib import Path
00003 from .file_copies_deleter import copies_deleter
00004
00005 from .normaliser import normalise
00006
00007 from .colors import G, R, N, Y
00008
00009
00010 EXT_FOLDER = {
00011     ("mp3", "ogg", "waw", "amr"): "audio",
00012     ("avi", "mp4", "mov", "mkv", "flv"): "video",
00013     ("jpeg", "png", "jpg", "svg"): "images",
00014     ("doc", "docx", "txt", "xlsx", "xls", "pptx"): "documents",
00015     ("djvu", "djv", "pdf", "tiff"): "books",
00016     ("zip", "gz", "tar", "7z"): "archives",
00017 }
00018
00019 """ ============================  ================================="""
00020
00021
00022 def get_file_cathegory(file: str):
00023     """         ."""
00024
00025     the_path = Path(file)
00026     ext = the_path.suffix.lstrip(".")
00027
00028     for exts in EXT_FOLDER.keys():
00029         if ext in exts:
00030             return EXT_FOLDER[exts]
00031     return None
00032
00033
00034 def normalise_file_name(file: str):
00035     """      normalise."""
00036
00037     the_path = Path(file)
00038     normalised_name = normalise(the_path.stem)
00039     new_file_path = the_path.parent.joinpath(
00040         "".join([normalised_name, the_path.suffix])
00041     )
00042     the_path.rename(new_file_path)
00043     return new_file_path
00044
00045
00046 def create_folders(root):
00047     """    root"""
00048
00049     for folder in EXT_FOLDER.values():
00050         Path(root).joinpath(folder).mkdir(exist_ok=True)
00051
00052
00053 def organize_files(
00054     path, level=0, known_exts=set(), unknown_exts=set(), categories=set()
00055 ):
00056     """     ."""
00057
00058     the_path = Path(path)
00059     if level == 0:
00060         global root_path
00061         root_path = the_path.resolve()
00062
00063     for item in the_path.iterdir():
00064         if item.is_dir() and item.name not in EXT_FOLDER.values():
00065             organize_files(item.resolve(), level + 1)
00066         else:
00067             category = get_file_cathegory(item.name)
00068             if category:
00069                 # -------------------------
00070                 categories.add(category)
00071                 known_exts.add(item.suffix)
00072                 item = normalise_file_name(item)
00073                 # -------------------------
00074                 root_name = Path(root_path).joinpath(category, item.name)
00075                 shutil.move(item, Path(root_path).joinpath(category))
00076                 if category == "archives":
00077                     unpack(
00078                         root_name,
00079                         Path(root_path).joinpath(category, item.stem),
00080                     )
00081             else:
00082                 if item.is_file():
```

```
00083                     unknown_exts.add(item.suffix)
00084     return tuple(unknown_exts)
00085
00086
00087 def unpack(archive_path, path_to_unpack):
00088     """ ."""
00089
00090     try:
00091         shutil.unpack_archive(archive_path, path_to_unpack)
00092     except OSError as er:
00093         print(er)
00094
00095
00096 def remove_empty(path):
00097     """  ."""
00098
00099     the_path = Path(path)
00100     empty = True
00101     for item in the_path.glob("*"):
00102         if item.is_file():
00103             empty = False
00104         if item.is_dir() and not remove_empty(item):
00105             empty = False
00106
00107     if empty:
00108         path.rmdir()
00109     return empty
00110
00111
00112 def known_exts(root):
00113     exts = set()
00114     for item in Path(root).iterdir():
00115         if item.is_dir():
00116             for file in item.iterdir():
00117                 if file.is_file():
00118                     exts.add(file.suffix)
00119     return exts
00120
00121
00122 """ ========================   ============================="""
00123
00124
00125 def sort_folder(root):
00126     path = Path(root)
00127     if not path.is_dir():
00128         return f"{Y}Warning!{N} The {root} is not a valid path!"
00129
00130     agreement = input(
00131         f"{Y}WARNING! {G}Are you sure you want to sort the files in CATALOG {Y + root}{N}? (y/n): "
00132     )
00133
00134     if agreement not in ("y", "Y", "yes", "Yes", "YES"):
00135         return f"{G}Operation approved!{N}"
00136
00137     copies_deleter(root)
00138     create_folders(root)
00139     unknown_exts = organize_files(root)
00140     remove_empty(root)
00141
00142     ke = ", ".join([ext for ext in known_exts(root)])
00143     print(f"{G}Known extensions:{N} {ke}")
00144
00145     if unknown_exts:
00146         ue = ", ".join([ext for ext in unknown_exts])
00147         print(f"{R}Unknown extensions:{N} {ue}")
00148
00149     for item in path.iterdir():
00150         if item.is_dir():
00151             num_of_files = sum(1 for file in item.iterdir() if file.is_file())
00152             print(
00153                 f"{G}Folder {Y}{item.name}{G} contain {Y}{num_of_files}{G} file(s){N}"
00154             )
00155
00156     return f"{G}Folder {Y}{root}{G} sorted!{N}"
```

8.15 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/normaliser.py File Reference

Namespaces

- namespace console_assistant

- namespace console_assistant.normaliser

Functions

- def console_assistant.normaliser.normalise (name)

8.16 normaliser.py

Go to the documentation of this file.

```
00001 from transliterate import translit
00002 from string import ascii_letters, digits
00003
00004
00005 def normalise(name):
00006     transliterated = translit(
00007         name, language_code="ru", reversed=True
00008     )  # from latin to cyrillic
00009     normalized_name = "".join(
00010         [
00011             char if char in ascii_letters or char in digits else "_"
00012             for char in transliterated
00013         ]
00014     )
00015     return normalized_name
00016
```

8.17 D:/Projects/Programming/Python/GoITHW/Console_↩ Assistant/console_assistant/notebook.py File Reference

Classes

- class console_assistant.notebook.Notebook

Namespaces

- namespace console_assistant
- namespace console_assistant.notebook

Variables

- namedtuple console_assistant.notebook.Note = namedtuple("Note", ["tags", "date", "text"])

8.18 notebook.py

Go to the documentation of this file.

```
00001 from collections import namedtuple, UserList
00002 from datetime import datetime
00003
00004 Note = namedtuple("Note", ["tags", "date", "text"])
00005
00006
00007 class Notebook(UserList):
00008     def update(self, notes):
00009         """ .     """
00010         self.data.clear()
00011         self.data.extend(notes)
00012
00013     def add_note(self, tags, note_text):
00014         """    """
00015
00016         note = Note(tags=tags, date=datetime.now(), text=note_text)
00017         self.data.append(note)
00018
00019     def remove_note(self, index):
00020         """ """
00021         self.data.pop(index)
00022
00023     def display_notes(self, tag=None, original_indices=False):
00024         """ ,    ,     original_indices=True"""
00025         notes = self.data
00026         if tag is not None:
00027             notes = [note for note in notes if tag in note.tags]
00028         if original_indices:
00029             notes = [
00030                 (note, index) for index, note in enumerate(self.data) if note in notes
00031             ]
00032         return notes
00033
00034     def iterator_notes(self, n: int = 10):
00035         """         n-."""
00036         items = self.data
00037         for i, note in enumerate(items):
00038             if i % n == 0:
00039                 items[i : i + n]
00040                 yield [(note, j) for j in range(i, i + n) if j < len(items)]
00041                 if i + n < len(items):
00042                     yield "continue"
00043
00044     def find_notes(self, search_term):
00045         """   """
00046         search_term = search_term.lower()
00047         results = []
00048         for i, note in enumerate(self.data):
00049             if search_term in note.text.lower():
00050                 results.append((note, i))
00051         return results
00052
00053     def sort_notes_by_tag(self):
00054         """   """
00055         return sorted(self.data, key=lambda note: tuple(note.tags))
00056
00057     def add_tag(self, index, tag):
00058         """     ."""
00059         note = self.data[index]
00060         note_tags = list(note.tags)
00061
00062         if tag not in note_tags:
00063             note_tags.append(tag)
00064             self.data[index] = note._replace(tags=tuple(note_tags))
00065
00066     def change_note(self, index, new_text):
00067         """  """
00068         note = self.data[index]
00069         self.data[index] = note._replace(text=new_text)
00070
00071     def __len__(self):
00072         return len(self.data)
00073
00074     def remove_tag(self, index, tag):
00075         """   """
00076         note = self.data[index]
00077         note_tags = list(note.tags)
00078         if tag in note_tags:
00079             note_tags.remove(tag)
00080             self.data[index] = note._replace(tags=tuple(note_tags))
00081             return True
00082         return False
```

8.19 D:/Projects/Programming/Python/GoITHW/Console_Assistant/console_assistant/serializer.py File Reference

Classes

- class console_assistant.serializer.Storage
- class console_assistant.serializer.PickleStorage

Namespaces

- namespace console_assistant
- namespace console_assistant.serializer

8.20 serializer.py

Go to the documentation of this file.

```
00001 """      """
00002
00003 import pickle
00004 from pathlib import Path
00005
00006
00007 class Storage:
00008     def export_file(object, filename: str):
00009         raise NotImplementedError
00010
00011     def import_file(object, filename: str):
00012         raise NotImplementedError
00013
00014
00015 class PickleStorage(Storage):
00016     @staticmethod
00017     def export_file(obj, filename):
00018         filename = Path(filename)
00019         with filename.open(mode="wb") as file:
00020             pickle.dump(obj, file)
00021
00022     @staticmethod
00023     def import_file(filename):
00024         filename = Path(filename)
00025         with filename.open(mode="rb") as file:
00026             return pickle.load(file)
00027
00028     @staticmethod
00029     def is_file_exist(filename):
00030         filename = Path(filename)
00031         if filename.exists():
00032             return True
00033         return False
```

8.21 D:/Projects/Programming/Python/GoITHW/Console_Assistant/README.MD File Reference

8.22 D:/Projects/Programming/Python/GoITHW/Console_Assistant/setup.py File Reference

Namespaces

- namespace setup

Variables

- setup.name
- setup.version
- setup.description
- setup.author
- setup.license
- setup.url
- setup.include_package_data
- setup.packages
- setup.entry_points
- setup.install_requires
- setup.package_data

8.23 setup.py

Go to the documentation of this file.

```
00001 from setuptools import setup, find_namespace_packages
00002
00003 setup(
00004     name="assistant",
00005     version="1.0.0",
00006     description="Command line assistant",
00007     author="sergiokapone",
00008     license="MIT",
00009     url="https://github.com/sergiokapone/console_assistant",
00010     include_package_data=True,
00011     packages=find_namespace_packages(),
00012     entry_points={"console_scripts": ["assistant = console_assistant.CLI:main"]},
00013     install_requires=[
00014         "prettytable==3.7.0",
00015         "Pygments",
00016         "transliterate",
00017         "prompt-toolkit>=3.0",
00018         "requests",
00019         "beautifulsoup4",
00020     ],
00021     package_data={
00022         "": ["README.md"],
00023     },
00024 )
```