**YourTelebot. Documentation**

Table of Contents

Introduction 2

Changing the Theme 2

Widgets 3

Variables 4

Functions 6

Tags 7

Project Structure 9

Attribution & License 10

Adding a New Language 11

Adding a Custom Command 12

Conclusion 17

**Introduction**

Welcome to the YourTelebot documentation. Our application is written in the Python programming language and utilizes numerous libraries and variables. This manual is designed to help you understand the code and customize it as needed.

To get started, we recommend installing a development environment if you haven’t already. Download Visual Studio Code, Sublime Text, PyCharm, or any other code editor of your choice. Avoid using the standard Notepad for working with Python, as it is not practical.

This documentation does not provide a detailed explanation of the entire code. It does not describe the purpose of every line or library. Most of what is not covered in this manual can be easily found online, is intuitively understandable, or is explained directly in the code through comments.

Additionally, this documentation assumes that you have basic knowledge of Python. You should understand variables, functions, dictionaries, lists, the purpose of some libraries, and other fundamental concepts.

The instructions provided here are only relevant for the [official version](https://github.com/qpikzz/YourTelebot) of the application.

Modification authors have the right to change the internal structure of the program. For user convenience, we recommend documenting any modifications.

Changing the Theme

This section will guide you on how to change the color of the desktop window.

1. Open YourTelebot.pyw using the previously installed code editor.
2. Press CTRL+F (Find) and type “Colors”. This will help you locate six variables (black, darkBlue, darkGreen... white). Inside the quotation marks, you can insert any color in HEX code format (e.g., #FFFFFF, #000013).

Changing the font is not recommended. The size of some widgets in tkinter is linked to the selected font, which may cause the application layout to break.

Widgets

In this section, we will explain the function of each element in the desktop version's window.

**Token** – Here, you need to insert the API token of the Telegram bot you created in BotFather (more details in the "Installation" section on the support website).

**Users** – In this field, you should insert the IDs of the accounts that will have access to your PC. More details are also available in the "Installation" section of the support website.

**Hide data under spoiler** – If this option is enabled, the bot will automatically place sensitive data under a spoiler. To view the hidden information, you will need to click on the concealed text.

**“2 Messages” Mode** – A working mode in which the bot deletes old messages, keeping only 2–3 messages in the chat. This can be useful for removing received data and keeping the chat uncluttered. Due to optimization constraints, it may not work perfectly, but if issues arise, you can fix them using the clearchat command.

**Save logs** – The program will create a log.txt file in the temporary folder (default: temp), where all user actions will be recorded.

**Language** – This setting allows you to choose the preferred language for the application. All available translations are located in the tl folder. Instructions for creating a custom language will be provided later.

**Auto–save files** – All photos, videos, voice messages, round videos, and documents sent to the bot will be saved in the temp folder.

**Console** – The largest widget in the right part of the program. As stated in the welcome message, it is designed to display various messages such as bot activation/deactivation and command usage. You can clear the console using the clear command.

**Enable/Disable** – A button that simply turns the bot on or off.

**Help** – Opens the support website, where you can find installation instructions, support contacts, frequently asked questions, and more.

Variables

Below, we describe the main variables used in the program.

**data** – The main variable, represented as a dictionary. It contains settings and all language configurations.

Structure:

|  |  |
| --- | --- |
| **Name** | **Type** |
| settings | Dictionary |
| lang1 | Dictionary |
| lang2 | Dictionary |
| langN | Dictionary |

Instead of lang1, lang2, ..., langN, the actual folder names containing translations will be used. In this case, these names are just examples.

Вы можете обратиться ко всем настройкам при помощи следующей конструкции:

|  |  |
| --- | --- |
| 1 | data["settings"] |

You can access all settings using the following syntax:

|  |  |
| --- | --- |
| 1 | print(data["settings"]) |

This will output a long string, which may be difficult to read. Let's break it down into a more understandable format:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Default Value** |
| language | str | “en” |
| token | str | “” |
| users | list (str) | [] |
| botWorked | bool | false |
| groups | bool | false |
| answerWithPic | bool | true |
| hide | bool | false |
| log | bool | false |
| 2mesMode | bool | false |
| autoSave | bool | false |
| tempFolder | str | “temp” |

Let's explain some points whose content may not be immediately obvious:

1. botWorked – A helper variable that determines whether the bot is running or not.
2. groups – A hidden setting that allows the bot to work in groups. We do not recommend enabling this, as an unauthorized person could join the chat and access sensitive information.
3. answerWithPic – Enables responses with images. This means the bot will attach a screenshot wherever it makes sense, such as in commands like write, click, etc.
4. hide – Enables spoiler–style hidden content.
5. autoSave – Automatically saves images, files, voice messages, and video messages sent to the bot.
6. tempFolder – The default temporary folder.

Here’s an example of viewing and modifying the hide setting:

|  |  |
| --- | --- |
| 1 | print(data["settings"]["hide"]) *# Output: False* |
| 2 | data["settings"]["hide"] = not data["settings"]["hide"] *# Toggle the value* |
| 3 | print(data["settings"]["hide"]) *# Output: True* |

All "keys" in `data` can be displayed using the code below. This will help you find out all available languages.

|  |  |
| --- | --- |
| 1 | print(data.keys()) |
| 2 | *# Output: dict\_keys(['settings', 'en', 'ru'])* |

Let's break down the structure of languages:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Brief Description** |
| error | dict | Error messages |
| log | dict | Messages written to log.txt |
| messages | dict | Responses sent by the bot |
| system | dict | Text of widgets, desktop window, etc. |

If you need to find a specific message or replace its value, open its corresponding file–category in the folder tl/(language)/(section).txt. To enable syntax highlighting in your code editor, set the language to "JSON".

**tk** – a variable that stores the program window. It supports all functions of the tkinter library. An example is shown below:

|  |  |
| --- | --- |
| 1 | tk["bg"] = black *# Set background color (black = #28262A)* |
| 2 | tk.title("YourTelebot") *# Set application title* |

**users, token, ..., twoMesMode** – widgets that store the current application settings. Of course, you can retrieve this data from data["settings"], but it's also useful to know how to access them directly.

|  |  |
| --- | --- |
| 1 | token.get() *# Retrieve the bot token as a string.* |
| 2 | users.get("1.0","end–1c").split("\n") *# Get the list of users.* |
| 3 | hideSpoiler.get() *# Get the boolean value of the "Hide under spoiler" setting.* |
| 4 | *# All similar variables: token, users, hideSpoiler, twoMesMode, logs, autoSave, language* |

**consoleMessage** – variable that stores the text to be displayed in the console. To properly handle everything (line breaks, tag insertion, etc.), it is more efficient to use the function `writeInConsole(text)`, which we will discuss later.

**user, text, chat** – auxiliary variables used in the functions start, text, and media to simplify the code. For example, instead of writing message.from\_user.id (to get the user ID), you can simply use user.id.

The meanings of the remaining variables should be intuitively clear.

Functions

This section explains the purpose of some functions. The arguments they take are listed in parentheses.

**writeInConsole(text)** – As mentioned earlier, this function is used to insert text into the console. It runs as a separate thread (independent of the main window). This is necessary to fix certain issues with the tkinter library.

**openData()** – A function used to create the data variable. It is responsible for reading settings and translations.

**answerWithPic(message, text)** – A function that allows the bot to respond with a screenshot. The message argument represents the message received by the bot. Here’s an example:

|  |  |
| --- | --- |
| 1 | bot=telebot.TeleBot(data["settings"]["token"],parse\_mode="html") |
| 2 | @bot.message\_handler(commands=["start"]) |
| 3 | def start(message): |
| 4 | answerWithPic(message, "text") |

**mode2mes(bot, message)** – A function that ensures the operation of the two–message mode. For optimization, it works by iterating through all messages from the last to the first and deleting them. If 7 consecutive messages have already been deleted, the function assumes that the chat has been cleared and stops further iteration.

**botFunc()** – like the console, is a separate thread. It contains the entire Telegram bot code. Also, this botFunc function has 3 nested functions: **start(message)**, **text(message)**, and **media(message)**. If you are familiar with the telebot module, their purpose is clear to you, but we will still briefly explain. The "start" function contains the code executed when the user sends the "/start" command. The "text" function contains the code executed when sending regular text messages. In match (an analog of switch–case from C–like languages), you can find almost all commands. A little later, we will look at how to create your own. The "media" function, as the name suggests, works with files, voice messages, photos, videos, and so on.

**botStartStop()** – a function whose name makes it clear that it is needed to switch the bot's state. In addition to changing the botWorking setting value, it changes the appearance of the Enable/Disable button and also outputs a message to the console.

**log(text, message)** – a function for saving logs. The "message" parameter is needed for tags (inserting the user's name and ID, text, and more).

**help()** – calls the help website. It is used to determine the theme - if the current time is before 6 AM or after 6 PM, the dark theme will be used; otherwise, the light theme will be displayed.

**saveSettings()** – automatic settings saving. The decision was made to remove the standard "Save" button to free up space in the window and improve user convenience. It runs as a separate thread.

**setTag(text, message)** – particularly attentive users may have noticed that, despite this function appearing at the beginning of the code, we explain it at the end of the section. This is because it is directly related to the next section. This function replaces application tags (written in square brackets) with HTML tags. This is necessary for formatting messages (bold, italics, etc.) and inserting certain variables into the text. If used outside the child functions of botFunc, "None" should be specified instead of message.

The remaining functions are auxiliary for widgets, and modifying them is not recommended, as it may break certain elements of the window. If you need an example of how to use a particular function, you can find it directly in the application's code.

Tags

While browsing translation files, you may have noticed strange constructions enclosed in square brackets. These are tags. They are used to optimize language development and insert variables. For example, instead of <blackquote> (quote), you can use [bq].

There are three types of tags:

1. **Permanent** – can be used anywhere in the code at any time.
2. **Dependent** – execution depends on a certain variable. For example, to use tags dependent on message, this variable must be specified in setTag.
3. **Local** – used only in a specific place.

Some tags have alternative versions. In other words, it doesn't matter whether you write [b] or [bold]; in any case, it will be replaced with <b>.

**Permanents:**

|  |  |  |
| --- | --- | --- |
| **Tag** | **Alt\*** | **Short Description** |
| b | bold | Bold text. **Like this.** |
| blackquote | bq | Quote. |
| code | cod | Converts text into code. Can be copied by clicking on the text. |
| date | – | Current date in the format "Day.Month.Year". |
| h | hide | If the user has enabled the "hide data under spoiler" setting, it will hide the enclosed content; otherwise, it works like [code]. |
| i | italic | Italic. *Like this.* |
| ip | – | Inserts the IP address of the computer running the program. |
| n | – | Line break. |
| now | – | Current time, accurate to milliseconds. |
| tempFolder | – | Current temporary directory. |
| time | – | Current time in the format "Hour:Min:Sec". |
| spoiler | sp | Hides text under a spoiler |
| u | underline | Underlined text. Like this. |
| x | – | Cursor position on the X-axis |
| y | – | Cursor position on the Y-axis |

\*Alt – short for Alternative.

**Dependent** **:**

|  |  |  |
| --- | --- | --- |
| **Tag** | **Var\*** | **Short description** |
| user\_username | message | User's nickname. Example: @durov |
| user\_id | message | User's ID. Example: 123456789 |
| user\_first\_name | message | User's first name as set in Telegram. |
| user\_last\_name | message | User's last name. |
| bot\_username | bot | Bot's nickname, whose token is specified in the application. |
| bot\_id | bot | Your bot's ID. |
| location | otherInfo | Location determined by IP. May not work stably. |
| org | otherInfo | Internet provider used on your computer. |
| timezone | otherInfo | Time zone determined by IP. |

\*Var – short for variable, on which the execution of the tag depends.

All other tags you encounter are local and most likely work only in that specific line.

**Adding a custom tag** – When expanding the bot’s capabilities, you may need to add your own tags.

If it is a permanent tag, press Ctrl+F and enter "Tags," then insert your modification into the replaces dictionary in JSON format ("[tag]": "<tag>").

If the update is dependent, insert the following construct after declaring the dictionary with permanent tags:

|  |  |
| --- | --- |
| 1 | if value: |
| 2 | replaces.update({ |
| 3 | "[Tag]":"<Tag>" |
| 4 | *# “[value]”:value* |
| 5 | }) |

Where value is the variable required for execution, [Tag] is your custom tag, and <Tag> is the new text or variable.

If your tag will only appear in one place, for example, in a specific command, you can use this method:

|  |  |
| --- | --- |
| 1 | *# Extracted from the "create" command:* |
| 2 | case "create": |
| 3 | with open(file,"w") as createFile: |
| 4 | bot.send\_message(chat.id,setTag(data[lang]["messages"]["create"],message).replace("[file]",file)) |
| 5 | *# [file] is replaced with the file path (e.g., temp/hello.txt).* |

Project Structure

Now we will explain in more detail where and what files are located.

The starting point is the root folder of the project. Contents:

1. yourTeleBot.pyw – the main file. It contains the core of the application.
2. data – data required for the program to function.
3. temp – temporary directory. It may have a different name or not exist at all.
4. tl – an abbreviation for "translates," meaning "translations." It stores all language files.

We move from top to bottom, going through each folder. Now, we navigate to the **data** folder.

1. **.ttf files** – Fonts. You should have downloaded them during the installation process.
2. **settings** – Stores the values of all settings described in the "Variables" section.
3. **splashes.txt** – A file containing all the desktop version's headers. These appear in the upper left corner when the application starts.
4. **images** – Contains images, mainly program icons and a few images used within the desktop version. Eye resolution: 20x13 pixels.

The contents of the **temp** folder are highly variable, but here’s what you might find:

1. **log.txt** – Logs with second-level accuracy, recording who used what.
2. **voice.mp3** – Speech synthesized using the "say" command.
3. **photo.jpg** – A photo taken from your webcam using the "photo" and "colorphoto" commands.
4. **Other files** – Saved media or files created with the "create" command that didn’t have a specified directory.

The contents of the **tl** folder consist of several subfolders. By default, these include en, none, and ru. As you might guess, each one contains a specific translation. Their structure is identical:

1. **dark.html** and **light.html** – Web pages for help documentation, opened locally in your browser.
2. **error.txt** – A file storing responses to various errors.
3. **log.txt** – Messages recorded in logs.
4. **manual.docx** – The document you are reading right now.
5. **messages.txt** – Bot responses depending on the command.
6. **system.txt** – Phrases used in the desktop interface.

It’s easy to notice that all .txt files actually use JSON format.

Attribution & License

You’ve probably already realized that YourTelebot is distributed under the **CC BY 4.0** license. Let’s break down what that means.

In short, you can:

1. Share (distribute) – Copy and distribute the material in any way and in any format, including for personal and commercial use.
2. Adapt (create derivative works) – Modify and create new content based on this material, including for commercial purposes..

PROVIDED THAT YOU COMPLY WITH THE FOLLOWING CONDITIONS:

1. Attribution – You must give appropriate credit. You can do this in any reasonable manner, but not in a way that suggests the creator of the application endorses you or your use of the work. We recommend including attribution in the welcome text of the console or on a webpage with instructions. A template for HTML (for web pages) and plain text will be provided later.
2. No additional restrictions – You may not impose legal terms or technological measures that legally restrict others from doing anything the license permits.

More details here: <https://creativecommons.org/licenses/by/4.0/>

**Attribution**

HTML – List changes in the second line (which is empty), in other words, in the <p> tag with id="changes".

|  |  |
| --- | --- |
| 1 | <div style="border: 1px solid #DADBDF; padding: 3rem 0; text–align: center; font–size: 1.5rem; color: #02000A; background–color: #E8E6EA"><p><a style="text–decoration: none; color: #2E5D42" href="https://github.com/YourTelebot">YourTelebot</a> © 2024 by <a style="text–decoration: none; color: #2E5D42" href="https://t.me/qpikzzbot">qpikzz</a> is licensed under <a style="text–decoration: none; color: #2E5D42" href="https://creativecommons.org/licenses/by/4.0/">CC BY 4.0</a></p><p style="font–size: 1rem;">differences from the original version:</p><p style="font–size: 1rem; color: #2E5D42;" id="changes"> |
| 2 |  |
| 3 | </p></div> |

Text – Can be inserted anywhere. We recommend placing it in system > consoleTitle. Replace [changes] with the changes you have made. We also suggest translating this text for all languages.

|  |  |
| --- | --- |
| 1 | [YourTelebot](https://github.com/YourTelebot) © 2024 by [qpikzz](https://t.me/qpikzzbot) is licensed under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Differences from the original version: [changes] |

Adding a New Language

The theory is over, now we can move on to real actions. Let's start by creating a new language.

1. Go to the tl directory and duplicate the none folder. This is a standard template where all fields are empty for easy filling. Rename the copied folder to the desired language name. It is recommended to use no more than 5–6 letters.
2. Open the newly created folder. To ensure you don't forget or mix anything up, we recommend opening an existing translation in parallel.
3. Fill in the .txt files. None of the lines should be left empty, as this may break the application.
4. Once you have filled in all the gaps, you can restart the application and select your language. If you do not plan to distribute your translation and do not want to deal with further modifications, this may be sufficient. Otherwise, proceed with editing the .html files, documentation, and user agreement.
5. To edit the help webpage, it is beneficial to have knowledge of HTML and CSS. Insert your translations inside the <body> tag within child elements like <a>, <p>, etc. We strongly recommend using existing versions as a reference.
6. The documentation only contains parts that are relevant for any language, such as tables, code snippets (without comments), and numbered lists. Use these as guidelines to add new sections.
7. The user agreement does not have any special characteristics—just rewrite it in your language. However, it is important not to alter its meaning. The original intent must be preserved.

Adding a Custom Command

Let's go through two examples: adding an "echo" command to our bot and creating a "random" command, which will be included in the final version of the program.

**Echo Command**

As you may have noticed, the original version of YourTeleBot does not include an echo function. This is because if a user accidentally finds our bot and does not receive a response beyond an echo, they are more likely to leave. However, for the sake of this example, let's go ahead and add it.

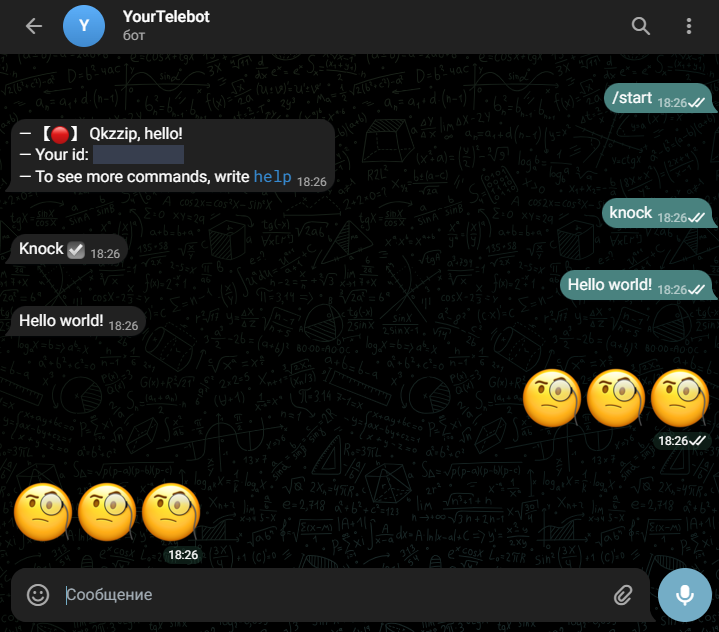
Simply put, if the bot does not recognize a command, it will return the same message to the user.

If the bot determined which command to execute using if/elif, we could just use else. However, YourTeleBot uses match. The alternative in this case is case \_:. We will add this after all existing commands.

Inside this case, we will instruct the bot to send the user's message back to them using the bot.send\_message function from the TeleBot library. The entire code takes up just two lines and looks like this:

|  |  |
| --- | --- |
| 1 | case \_: |
| 2 | bot.send\_message(chat.id, text) |

Now, let's test the code.



Great! Everything works!

Now, let's modify the behavior so that instead of repeating the user's message, the bot responds with a notification stating that it does not understand the command.

Since this is a message sent by the bot, it is recommended to store it in messages.txt for all available translations. Add the following line:

|  |  |
| --- | --- |
| 1 | "whachaMean":"What does this mean??? I don't know such a command :(" |

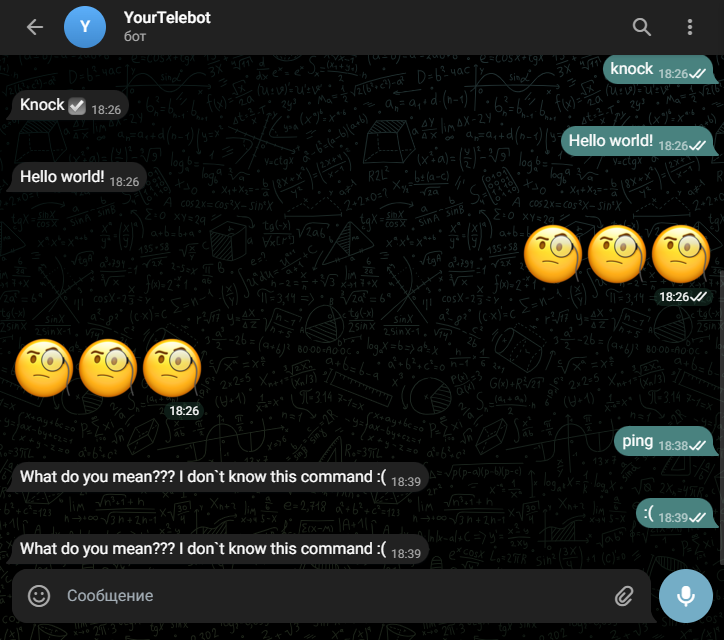
Inside bot.send\_message, replace text with the setTag function, passing in our newly added message.

You might wonder why we are using setTag when we are not currently using tags. This is done as a precaution in case you later decide to add formatting or insert variables into the text.

Here is the updated code:

|  |  |
| --- | --- |
| 1 | case \_: |
| 2 | bot.send\_message(chat.id,setTag(data[lang]["messages"]["whachaMean"], None)) |

Restart the bot and test the changes in the same chat:



**Random**

Let's create a command that will return a random number. By default, if the user has not provided any additional numbers, it will generate a number in the range from 0 to 100. If one number is specified (for example: random 10), the bot will return a random number from 0 to 10. When two numbers are specified, a random number is returned within the given range. Example: if you specify 1000 and 200, a number will be chosen in the range from 200 to 1000.

First, let's add the necessary phrases to the translations in messages.txt.

|  |  |
| --- | --- |
| 1 | "randomDefault":"Random number from 0 to 100: [cod][rand][/cod]" |
| 2 | "randomWithA":"Random number from 0 to [A]: [cod][rand][/cod]" |
| 3 | "randomWithAB":"Random number from [A] to [B]: [cod][rand][/cod]" |

The first message will be used when the user does not specify any numbers.

The second will be sent in response to a message with one parameter.

The third, obviously, will be used when both range limits are specified.

Important note: The [B] tag must be written in uppercase. If written in lowercase, it will cause an error because it will be replaced with <b> (the opening tag for bold text).

Let's add a new "case" to the match, where we'll create a list of numbers provided by the user:

|  |  |
| --- | --- |
| 1 | case "random": |
| 2 | rand = text.split() *# Create a list of all words in the message* |
| 3 | rand.pop(0) *# Remove the word "random"* |
| 4 | rand = list(map(int, rand)) *# Convert all elements from str to int* |

Next, we need to check how many elements are in the rand list, as this will determine the response.

First, let's handle the case where no numbers are provided:

|  |  |
| --- | --- |
| 1 | if not rand: |
| 2 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomDefault"], None).replace("[rand]",str(r.randint(0,100)))) |

The second line might seem complicated, so let's break it down. First, the randomDefault message from messages.txt is processed with setTag, inserting any constant tags. Then, the local tag [rand] is replaced with a random number between 0 and 100. Finally, the processed message is sent to the user.

Similarly, we handle the case where the user provides only one number.

The logic is simple: if you have a box with one ball and are asked to pick a random ball from it, it will obviously be that one ball.

|  |  |
| --- | --- |
| 1 | elif len(rand) == 1: |
| 2 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithA"], None).replace("[rand]",str(r.randint(min(0,rand[0]),max(0,rand[0])))).replace("[A]",str(rand[0]))) |

The key difference from the "default" case is the local tag [A], which gets replaced with the number provided by the user. Later, we'll introduce a similar [B] tag.

Let's add a condition for cases where the user enters a different number of digits. Of course, we will only use two, but there's no need to limit the message length—any extra numbers will simply be ignored.

|  |  |
| --- | --- |
| 1 | else: |
| 2 | if rand[0] == rand[1]: |
| 3 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithAB"], None).replace("[rand]",f"{rand[0]} ¯\\\_(ツ)\_/¯").replace("[A]",str(rand[0])).replace("[B]",str(rand[1]))) |
| 4 | else: |
| 5 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithAB"], None).replace("[rand]",str(r.randint(min(rand[0],rand[1]),max(rand[0],rand[1])))).replace("[A]",str(min(rand[0],rand[1]))).replace("[B]",str(max(rand[0],rand[1])))) |

Don't forget to use min() and max(). In the case of tags [A] and [B], this is purely for aesthetics, but in randint, it can cause an error since the first parameter must always be the smaller value, followed by the larger one.

We also need to account for people who, when asked "how many?", respond with "a lot." If letters are entered instead of numbers, our program will crash. To prevent this, we will wrap the code in a try-except block to catch a ValueError.

Let's add an additional phrase for this in error.txt

|  |  |
| --- | --- |
| 1 | "numberRandomError":"You did not specify a number![n]Example: [cod]random 1 10[/cod]" |

Весь код команды выглядит вот так:

|  |  |
| --- | --- |
| 1 | case "random": |
| 2 | try: |
| 3 | rand = text.split() *# Create a list of all words in the message* |
| 4 | rand.pop(0) *# Remove the word "random"* |
| 5 | rand = list(map(int, rand)) *# Convert all elements from str to int* |
| 6 |  |
| 7 | if not rand: |
| 8 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomDefault"], None).replace("[rand]",str(r.randint(0,100)))) |
| 9 |  |
| 10 | elif len(rand) == 1: |
| 11 | if rand[0] == 0: |
| 12 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithA"], None).replace("[rand]","0 ¯\\\_(ツ)\_/¯").replace("[A]","0")) |
| 13 | else: |
| 14 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithA"], None).replace("[rand]",str(r.randint(min(0,rand[0]),max(0,rand[0])))).replace("[A]",str(rand[0]))) |
| 15 |  |
| 16 | else: |
| 17 | if rand[0] == rand[1]: |
| 18 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithAB"], None).replace("[rand]",f"{rand[0]} ¯\\\_(ツ)\_/¯").replace("[A]",str(rand[0])).replace("[B]",str(rand[1]))) |
| 19 | else: |
| 20 | bot.send\_message(chat.id, setTag(data[lang]["messages"]["randomWithAB"], None).replace("[rand]",str(r.randint(min(rand[0],rand[1]),max(rand[0],rand[1])))).replace("[A]",str(min(rand[0],rand[1]))).replace("[B]",str(max(rand[0],rand[1])))) |
| 21 |  |
| 22 | except ValueError: *# Выполняет нижеуказанный код в случае, если ошибка была найдена* |
| 23 | bot.send\_message(chat.id, setTag(data[lang]["error"]["numberRandomError"], None)) |

Conclusion

This concludes the documentation. We hope you found what you were looking for. If you still have questions, try searching for answers online or using neural networks. In extreme cases, you can contact support.

Take care! By qpikzz.

Author • Source Code • User Agreement