Onboarding Chatbot Documentation

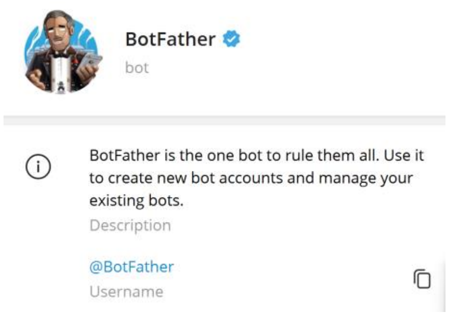
The Smart Contracts Lab Telegram chatbot was developed as an innovative solution to enhance communication during the onboarding process of writing a bachelor or master thesis at the University of Zurich. The onboarding bot aims to provide new students with easy access to essential information and a platform to ask questions, ensuring a smooth and informed entry into the program.

The Onboarding Chatbot is additionally documented on GitHub:
https://github.com/SCL-Project/Operations/tree/main/OnboardingBot

1. Initial Setup

1. Creating a Bot:

   - Start a chat with BotFather on Telegram

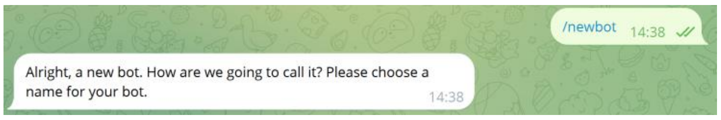


BotFather
bot

BotFather is the one bot to rule them all. Use it to create new bot accounts and manage your existing bots.
Description

@BotFather
Username

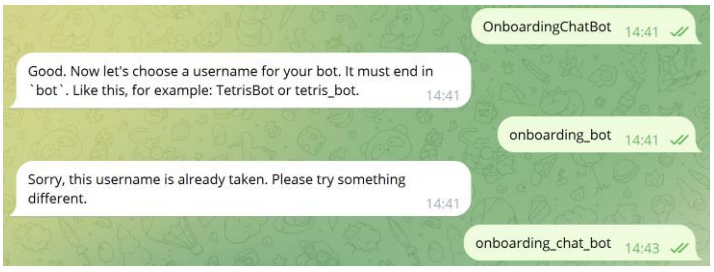
   - Use the /newbot command to create a new bot



/newbot 14:38

Alright, a new bot. How are we going to call it? Please choose a name for your bot. 14:38

   - Follow the prompts to set a name and a username for the bot



OnboardingChatBot 14:41

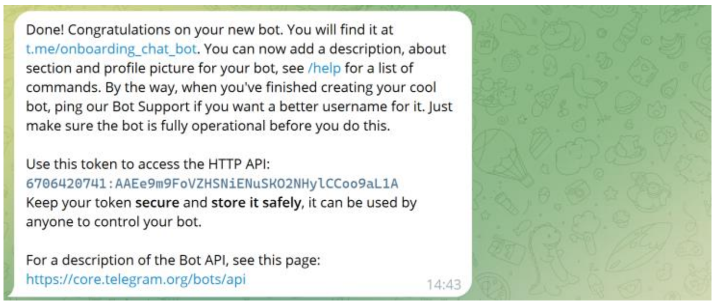
Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot. 14:41

onboarding_bot 14:41

Sorry, this username is already taken. Please try something different. 14:41

onboarding_chat_bot 14:43

- Receive a Token: After the setup, the BotFather will provide a unique token, which is used to connect with the Telegram API



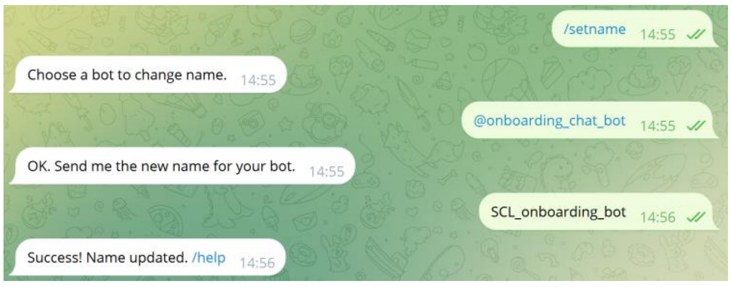
Done! Congratulations on your new bot. You will find it at t.me/onboarding_chat_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
`6706420741:AAEe9m9FoVZHSNiENuSKO2NHylCCoo9aL1A`
Keep your token **secure** and **store it safely**, it can be used by anyone to control your bot.

For a description of the Bot API, see this page:
https://core.telegram.org/bots/api 14:43

2. Navigate to your new bot and start configuring the Bot by customizing features using the following commands:

- /setname: update the name



/setname 14:55

Choose a bot to change name. 14:55

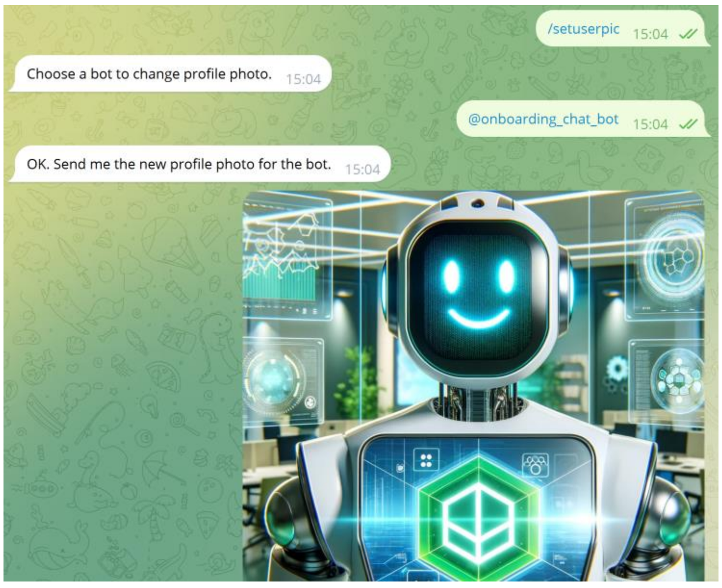
@onboarding_chat_bot 14:55

OK. Send me the new name for your bot. 14:55

SCL_onboarding_bot 14:56

Success! Name updated. /help 14:56

- /setuserpic: With the use of DALL-E 3 we used an AI generated profile picture for our bot



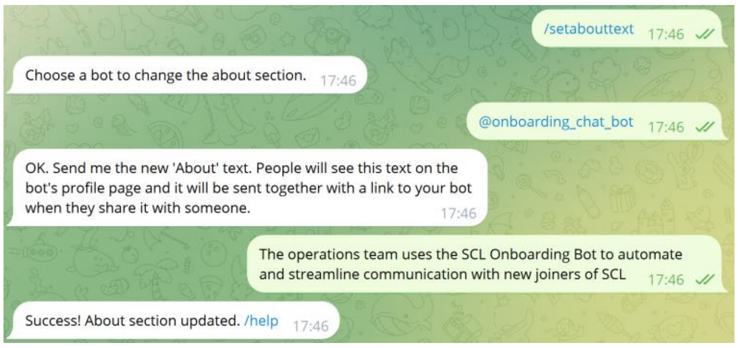
/setuserpic 15:04

Choose a bot to change profile photo. 15:04

@onboarding_chat_bot 15:04

OK. Send me the new profile photo for the bot. 15:04

- /setabouttext: short text that people will see on the bot's profile page and it will be sent together with a link to your bot when they share it with someone:
  “The operations team uses the SCL Onboarding Bot to automate and streamline communication with new joiners of SCL”



/setabouttext 17:46

Choose a bot to change the about section. 17:46

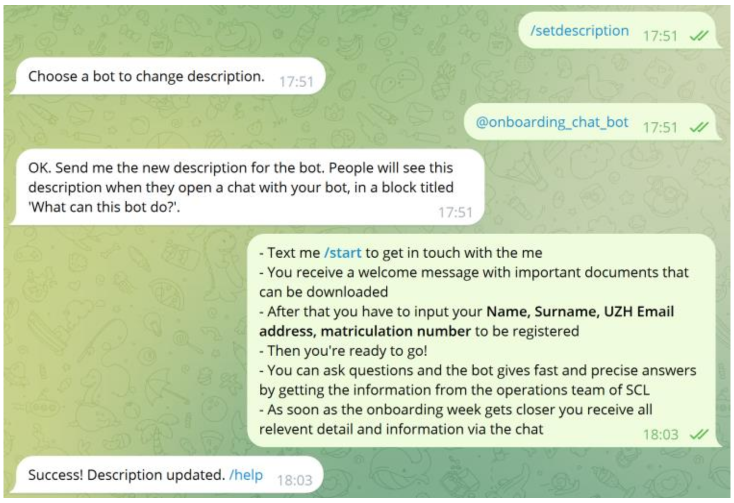
@onboarding_chat_bot 17:46

OK. Send me the new 'About' text. People will see this text on the bot's profile page and it will be sent together with a link to your bot when they share it with someone. 17:46

The operations team uses the SCL Onboarding Bot to automate and streamline communication with new joiners of SCL 17:46

Success! About section updated. /help 17:46

- /setdescription: Contains the information for the users on how to use the bot



/setdescription 17:51

Choose a bot to change description. 17:51

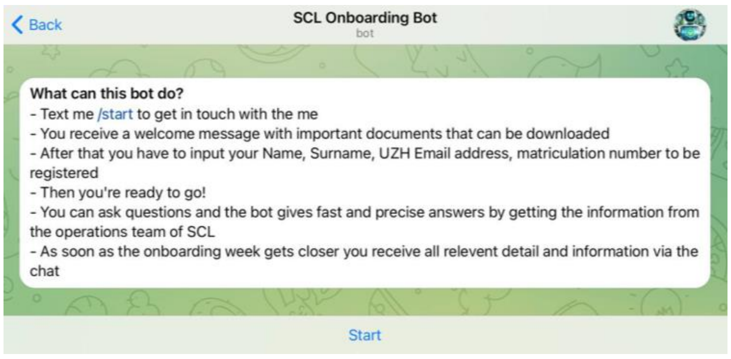
@onboarding_chat_bot 17:51

OK. Send me the new description for the bot. People will see this description when they open a chat with your bot, in a block titled 'What can this bot do?'. 17:51

- Text me /start to get in touch with the me
- You receive a welcome message with important documents that can be downloaded
- After that you have to input your **Name, Surname, UZH Email address, matriculation number** to be registered
- Then you're ready to go!
- You can ask questions and the bot gives fast and precise answers by getting the information from the operations team of SCL
- As soon as the onboarding week gets closer you receive all relevent detail and information via the chat 18:03

Success! Description updated. /help 18:03

3. Finally the bot will look like this:



What can this bot do?
- Text me /start to get in touch with the me
- You receive a welcome message with important documents that can be downloaded
- After that you have to input your Name, Surname, UZH Email address, matriculation number to be registered
- Then you're ready to go!
- You can ask questions and the bot gives fast and precise answers by getting the information from the operations team of SCL
- As soon as the onboarding week gets closer you receive all relevent detail and information via the chat

2. Overview of the Python Script

At its core, the script is a set of instructions that tells the chatbot how to interact with users and handle various tasks. When the script runs, it connects the chatbot to Telegram's services, allowing it to send and receive messages. The primary function of the chatbot is to assist new students during their onboarding process. It provides important resources such as the onboarding schedule and welcome brochure. Additionally, the bot answers questions that the students ask. The chatbot starts by greeting new users and offering them quick access to important documents. It can send files, such as a PDF of the welcome brochure or an image of the onboarding schedule, directly to the students through Telegram. The users of the bot can register themselves with providing their name, surname, UZH email address and matriculation number. Then they simply can ask questions that the onboarding task force is going to answer via the chatbot.

One of the critical features of this chatbot is its ability to manage user interactions effectively. It keeps track of which students have already registered, avoiding duplicate registrations. Whenever a new student sends a message, the chatbot records their details and notifies the project's task force in the defined group chat, ensuring that all questions from the students are getting their response. For the task force, the chatbot offers functionalities like sending replies to specific students or broadcasting messages to all registered users. This is particularly useful for sharing updates or important information with everyone at once.

3. Setting Up Key Data

To ensure the chatbot functions correctly, certain key pieces of information must be set up in the Python script. These include the Bot Token, Group Chat ID, and file paths for storing and accessing data:

```
# Bot Token from BotFather
TOKEN = '6706420741:AAEe9m9FoVZHSNiENuSKO2NHylCCoo9aL1A'

# Group Chat ID of the Onboarding Task force
TASK FORCE_CHAT_ID = -1001998746458

# CSV file path for storing user data
CSV_FILE = 'OnboardingBot/confirmed_users.csv'

# File paths
SCHEDULE_PATH = 'OnboardingBot/Schedule.png'
BROCHURE_PATH = 'OnboardingBot/Welcome Brochure.pdf'
```

4. Main Libraries Used

The Telegram chatbot for the Smart Contracts Lab project utilizes four main Python libraries:

- Telegram: Required to update queries as well as keyboard buttons and markups
- Telegram.ext: Handles communication with Telegram's Bot API, enabling the bot to receive and send messages, and manage user interactions.
- csv: Manages reading and writing of CSV files, used for storing and accessing user data.
- logging: Records the chatbot's activities and errors, aiding in monitoring and troubleshooting.

These libraries are fundamental to the chatbot's operation, ensuring efficient user interaction and reliable performance.

5. Function Descriptions

Function: start

- Purpose: Initiate interaction with a new user.
- Operation: Displays a welcome message and options to access the onboarding schedule and brochure.

```
async def start(update: Update, context: CallbackContext):
    keyboard = [
        [InlineKeyboardButton("Get Onboarding Schedule", callback_data="get_schedule")],
        [InlineKeyboardButton("Get Welcome Brochure", callback_data="get_brochure")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.message.reply_text(
        "Hello and Welcome to Smart Contracts Lab Onboarding Bot,\n\n"
        "This bot is here to help you with your onboarding process.\n"
        "Before we can start please input your name, surname, UZH email address and matriculation number. "
        "You will get another message when this Chat is ready to use.\n\n"
        "Further informations will be provided when the onboarding week gets closer"
        "You can also get the Onboarding Schedule and Welcome Brochure below.\n\n"
        "Thank you.\n\nBest regards,\nThe Onboarding Team",
        reply_markup=reply_markup
    )
```

Function: send_schedule and send_brochure

- Purpose: Provide onboarding schedule and welcome brochure.
- Operation: Sends a photo of the schedule and a document of the brochure to the user.

```
async def send_schedule(update: Update, context: CallbackContext):
    query = update.callback_query
    await query.answer()
    with open(SCHEDULE_PATH, 'rb') as file:
        await context.bot.send_photo(chat_id=query.message.chat_id, photo=file)

async def send_brochure(update: Update, context: CallbackContext):
    query = update.callback_query
    await query.answer()
    with open(BROCHURE_PATH, 'rb') as file:
        await context.bot.send_document(chat_id=query.message.chat_id, document=file)
```

Function: load_registered_users

- Purpose: Load the list of registered users from a CSV file.
- Operation: Reads the CSV file and returns a set of registered user IDs.

```
def load_registered_users(csv_file):
    user_ids = set()
    try:
        with open(csv_file, mode='r', newline='') as file:
            reader = csv.reader(file)
            for row in reader:
                if row:  # Check if row is not empty
                    user_id = row[0]  # Assuming the user ID is the first column
                    user_ids.add(int(user_id))
    except FileNotFoundError:
        logger.error(f"File not found: {csv_file}")
    except Exception as e:
        logger.error(f"Error reading CSV file: {e}")

    return list(user_ids)
```

Function: register_user

- Purpose: Register a new user.
- Operation: Adds a new user's information to the CSV file.

```
def is_user_registered(user_id, csv_file):
    with open(csv_file, mode='r', newline='') as file:
        reader = csv.reader(file)
        for row in reader:
            if row and row[0] == str(user_id):
                return True
    return False
```

Function: handle_message

- Purpose: Handle incoming messages from users.
- Operation: Registers new users or forwards their messages to the task force.

```
async def handle_message(update: Update, context: CallbackContext):
    user = update.effective_user
    if update.message.text:
        # Check if user is already registered
        if not is_user_registered(user.id, CSV_FILE):
            # New user: save user's message to CSV and notify the task force for confirmation
            with open(CSV_FILE, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow([user.id, user.first_name, user.last_name, update.message.text])

            keyboard = [[InlineKeyboardButton("Confirm", callback_data=f'confirm_{user.id}')]]
            reply_markup = InlineKeyboardMarkup(keyboard)
            await context.bot.send_message(chat_id=TASK FORCE_CHAT_ID,
                                           text=f"New joiner details:\n\n{update.message.text}\n\nConfirm the user?",
                                           reply_markup=reply_markup)
        else:
            # Existing user: forward their message to the task force
            await context.bot.send_message(chat_id=TASK FORCE_CHAT_ID,
                                          text=f"Message from user {user.id} ({user.first_name}
{user.last_name}):\n\n{update.message.text}")
```

Function: confirm_user

- Purpose: Confirm the registration of a new user.
- Operation: Sends a confirmation message to the user.

```
async def confirm_user(update: Update, context: CallbackContext):
    query = update.callback_query
    await query.answer()

    user_id = query.data.split('_')[1]
    bot = context.bot
    await bot.send_message(chat_id=user_id,
                           text="Hi again,\n\nYou are successfully registered and you can now use this chat"
                                "\nWe will send you all the information on a later date.\n"
                                "If you have any questions, do not hesitate to write them here and we will get back to you.\n\n"
                                "Best regards,\nThe Onboarding Team")
```

Function: reply_to_user

- Purpose: Enable task force to reply to a specific user.
- Operation: Sends a message from the task force to a specified user.

```
async def reply_to_user(update: Update, context: CallbackContext):
    # Ensure that this command is used in the TASK FORCE_CHAT_ID only
    if update.message.chat_id != TASK FORCE_CHAT_ID:
        return

    try:
        args = context.args  # Extract arguments passed with the command
        if len(args) < 2:
            raise ValueError("Insufficient arguments.")

        user_id = int(args[0])  # The first argument is the user ID
        message_to_send = ' '.join(args[1:])  # The rest is the message

        await context.bot.send_message(chat_id=user_id, text=message_to_send)
        await update.message.reply_text(f"Message sent to user {user_id}")
    except (IndexError, ValueError, TypeError) as e:
        await update.message.reply_text("Usage: /reply USER_ID MESSAGE")
```

Function: broadcast_to_users

- Purpose: Send a broadcast message to all registered users.
- Operation: Distributes a message to every user listed in the CSV file.

```
async def broadcast_to_users(update: Update, context: CallbackContext):
    if update.message.chat_id != TASK FORCE_CHAT_ID:
        return

    try:
        message_to_broadcast = ' '.join(context.args)
        if not message_to_broadcast:
            raise ValueError("No message provided.")

        # Load the registered users from the CSV file
        registered_users = load_registered_users(CSV_FILE)

        # Send the broadcast message to each registered user
        for user_id in registered_users:
            await context.bot.send_message(chat_id=user_id, text=message_to_broadcast)

        await update.message.reply_text(f"Broadcast message sent to {len(registered_users)} users.")
    except ValueError as e:
        await update.message.reply_text("Usage: /broadcast MESSAGE")
```

Function: error

- Purpose: Log errors caused by updates.
- Operation: Records errors in the logger for debugging.

```
def error(update: Update, context: CallbackContext):
    """Log Errors caused by Updates."""
    logger.warning('Update "%s" caused error "%s"', update, context.error)
```

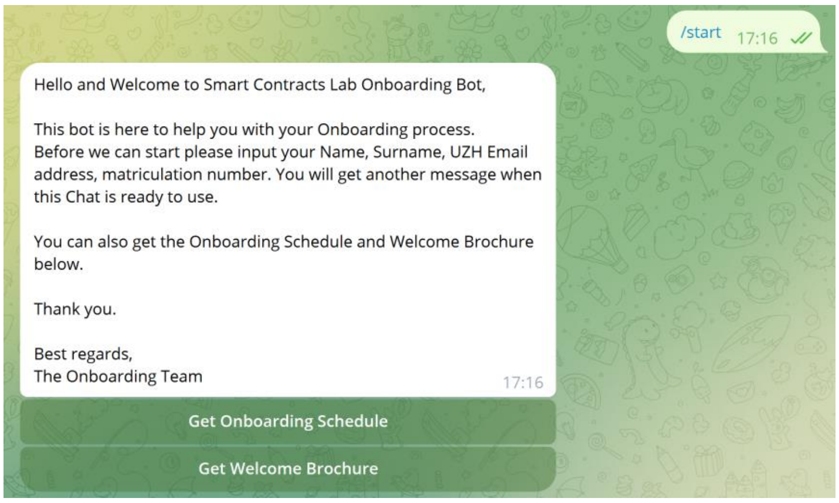
Function: main

- Purpose: Main entry point of the script.
- Operation: Sets up the handlers and starts the bot.

```
def main() -> None:
    application = Application.builder().token(TOKEN).build()

    start_handler = CommandHandler("start", start)
    message_handler = MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message)
    confirm_handler = CallbackQueryHandler(confirm_user, pattern='^confirm_')
    schedule_handler = CallbackQueryHandler(send_schedule, pattern='^get_schedule$')
    brochure_handler = CallbackQueryHandler(send_brochure, pattern='^get_brochure$')
    reply_handler = CommandHandler("reply", reply_to_user, filters.Chat(chat_id=TASK FORCE_CHAT_ID))
    broadcast_handler = CommandHandler("broadcast", broadcast_to_users, filters.Chat(chat_id=TASK
FORCE_CHAT_ID))
    confirm_handler = CallbackQueryHandler(confirm_user, pattern='^confirm_')

    application.add_handler(confirm_handler)
    application.add_handler(broadcast_handler)
    application.add_handler(start_handler)
    application.add_handler(message_handler)
    application.add_handler(confirm_handler)
    application.add_handler(schedule_handler)
    application.add_handler(brochure_handler)
    application.add_handler(reply_handler)
    application.add_error_handler(error)

    application.run_polling()
```

6. Using the Chatbot

1. Starting the Bot: Run the script to start the bot (if not hosted on a server).
2. Interacting with Users: Users can start interacting by searching «onboarding_chat_bot» on Telegram (or directly: https://t.me/onboarding_chat_bot) and either press the start button or type «/start». They receive a welcome message including keyboards to download the relevant documents for the onboarding week.



/start 17:16

Hello and Welcome to Smart Contracts Lab Onboarding Bot,

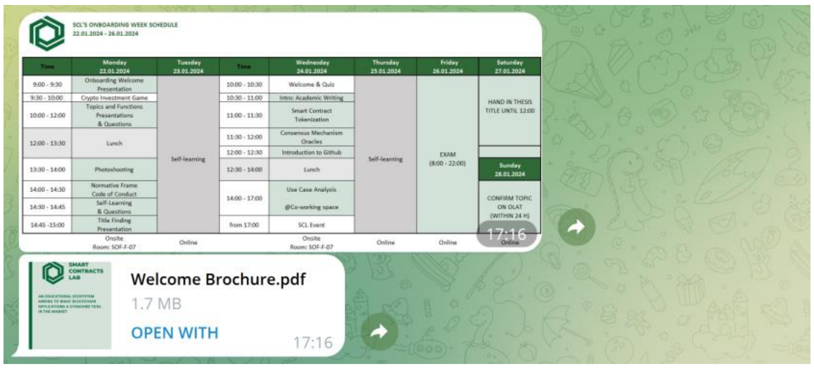
This bot is here to help you with your Onboarding process.
Before we can start please input your Name, Surname, UZH Email address, matriculation number. You will get another message when this Chat is ready to use.

You can also get the Onboarding Schedule and Welcome Brochure below.

Thank you.

Best regards,
The Onboarding Team 17:16

3. After pressing the buttons they get sent the documents via the Telegram chat.



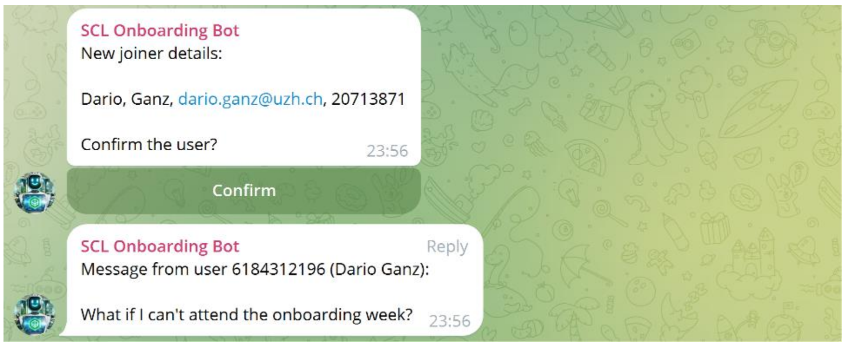
SCL'S ONBOARDING WEEK SCHEDULE
22.01.2024 - 26.01.2024

Time	Monday 22.01.2024	Tuesday 23.01.2024	Time	Wednesday 24.01.2024	Thursday 25.01.2024	Friday 26.01.2024	Saturday 27.01.2024
9:00 - 9:30	Onboarding Welcome Presentation	Self-learning	10:00 - 10:30	Welcome & Quiz	Self-learning	EXAM (8:00 - 22:00)	HAND IN THESIS TITLE UNTIL 12:00
9:30 - 10:00	Crypto Investment Game		10:30 - 11:00	Intro: Academic Writing			
10:00 - 12:00	Topics and Functions Presentations & Questions		11:00 - 11:30	Smart Contract Tokenization			
12:00 - 13:30	Lunch		11:30 - 12:00	Consensus Mechanism Oracles			
			12:00 - 12:30	Introduction to Github			
13:30 - 14:00	Photoshooting		12:30 - 14:00	Lunch			Sunday 28.01.2024
14:00 - 14:30	Normative Frame Code of Conduct		14:00 - 17:00	Use Case Analysis @Co-working space			CONFIRM TOPIC ON OLAT (WITHIN 24 H)
14:30 - 14:45	Self-Learning & Questions						
14:45 -15:00	Title Finding Presentation		from 17:00	SCL Event			
	Onsite Room: SOF-F-07	Online		Onsite Room: SOF-F-07	Online	Online	Online

17:16

Welcome Brochure.pdf
1.7 MB
OPEN WITH 17:16

4. Users of the Onboarding Chatbot can register themselves with providing the following information:
   "name, surname, UZH email address and matriculation number"
   The onboarding task force can manage this data of the registered users by accessing the «confirmed_users.csv».
   After they sent these information, the Onboarding Chatbot texts the onboarding task force group chat and the user can be confirmed. When the user is confirmed, questions can be asked in the chat and the Chatbot forwards these into the group chat with providing the USER_ID and name of the sender of the question.



SCL Onboarding Bot
New joiner details:

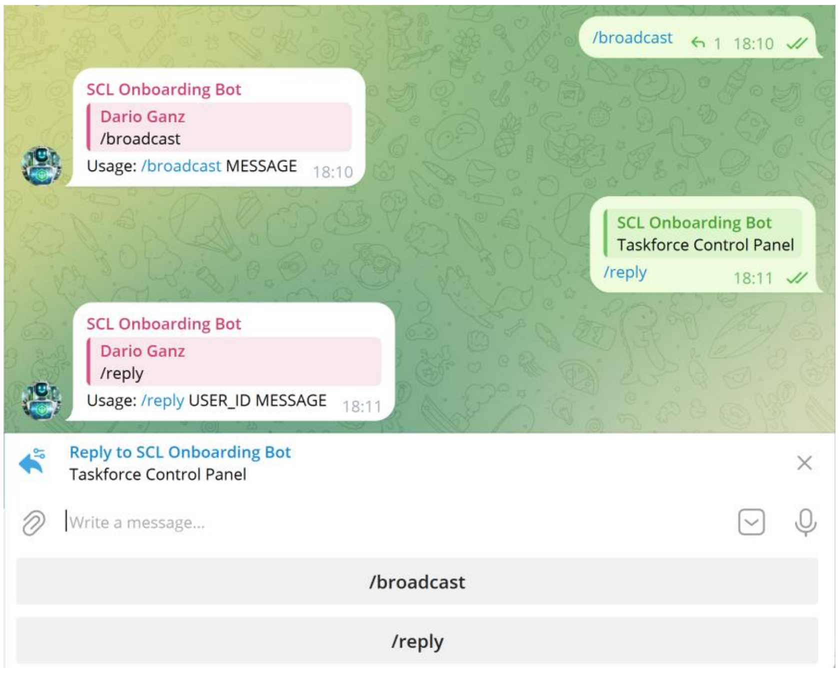
Dario, Ganz, dario.ganz@uzh.ch, 20713871

Confirm the user? 23:56

SCL Onboarding Bot
Message from user 6184312196 (Dario Ganz):

What if I can't attend the onboarding week? 23:56

5. Task force Management: The task force can use command buttons /reply and /broadcast to manage communication. By pressing **/reply**, the message is sent to a specific user with the use of his/her USER_ID. When using **/broadcast**, the message gets sent to all users that have registered to the Onboarding Chatbot.



/broadcast 1 18:10

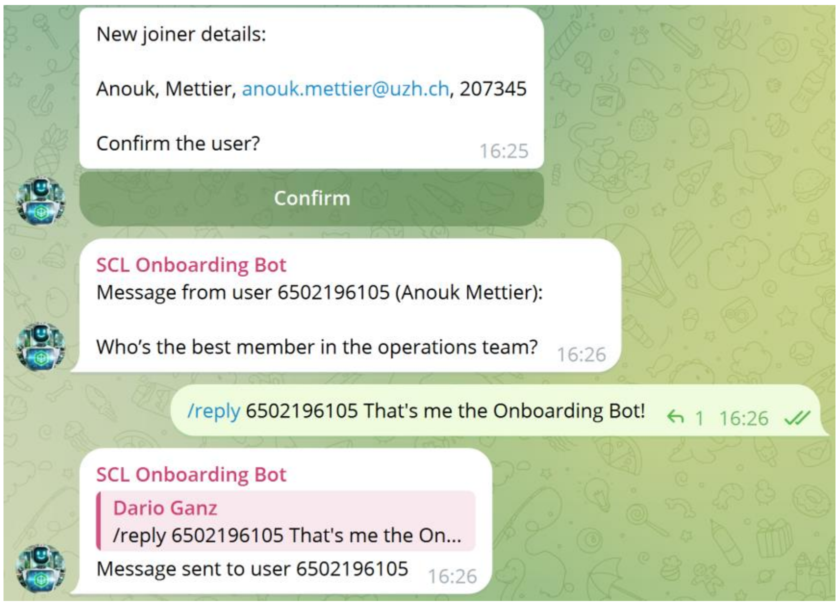
SCL Onboarding Bot
Dario Ganz
/broadcast
Usage: /broadcast MESSAGE 18:10

SCL Onboarding Bot
Taskforce Control Panel
/reply 18:11

SCL Onboarding Bot
Dario Ganz
/reply
Usage: /reply USER_ID MESSAGE 18:11

Reply to SCL Onboarding Bot
Taskforce Control Panel

6. Here is another example of a newly registered user. With the /reply function the Onboarding Chatbot told her who the best member of the Operations team is. After sending the message with **/reply USER_ID MESSAGE**, the Chatbot sends a confirmation message that the message was sent.



New joiner details:

Anouk, Mettier, anouk.mettier@uzh.ch, 207345

Confirm the user? 16:25

Confirm

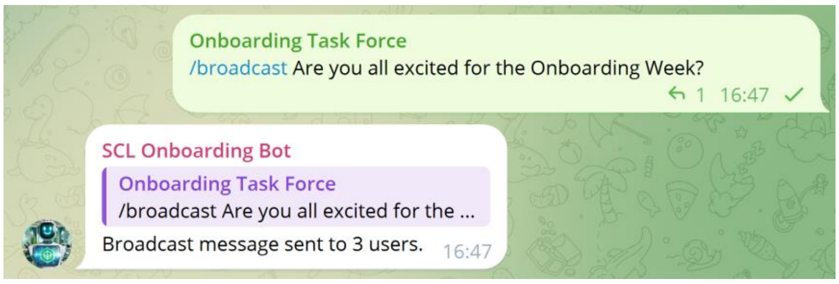
SCL Onboarding Bot
Message from user 6502196105 (Anouk Mettier):

Who's the best member in the operations team? 16:26

/reply 6502196105 That's me the Onboarding Bot! 1 16:26

SCL Onboarding Bot
Dario Ganz
/reply 6502196105 That's me the On...
Message sent to user 6502196105 16:26

7. If I **/broadcast** a message the Chatbot gives you a confirmation with the information about to how many users he sent the message to.

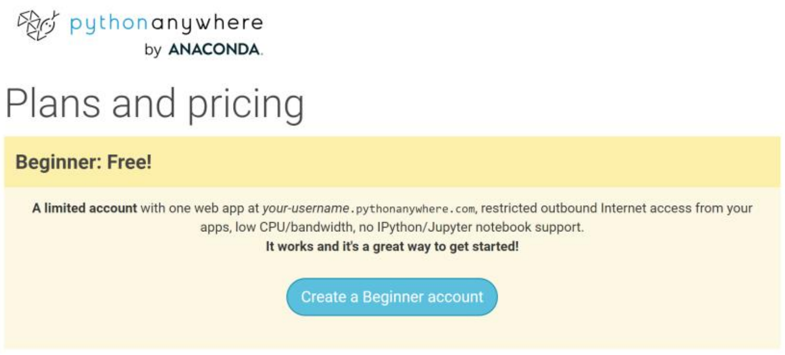


Onboarding Task Force
/broadcast Are you all excited for the Onboarding Week?
1 16:47

SCL Onboarding Bot
Onboarding Task Force
/broadcast Are you all excited for the ...
Broadcast message sent to 3 users. 16:47

7. Hosting on PythonAnywhere for Continuous Code Execution

- Create a freee beginner account on PythonAnywhere

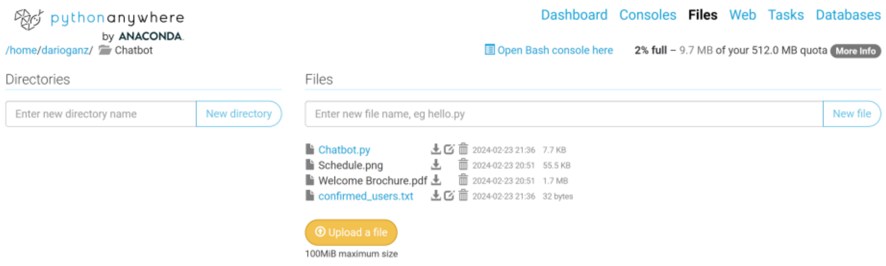


Plans and pricing

Beginner: Free!

A limited account with one web app at *your-username*.`pythonanywhere.com`, restricted outbound Internet access from your apps, low CPU/bandwidth, no IPython/Jupyter notebook support.
It works and it's a great way to get started!

- Create a new repository and upload all relevant files to execute the bot

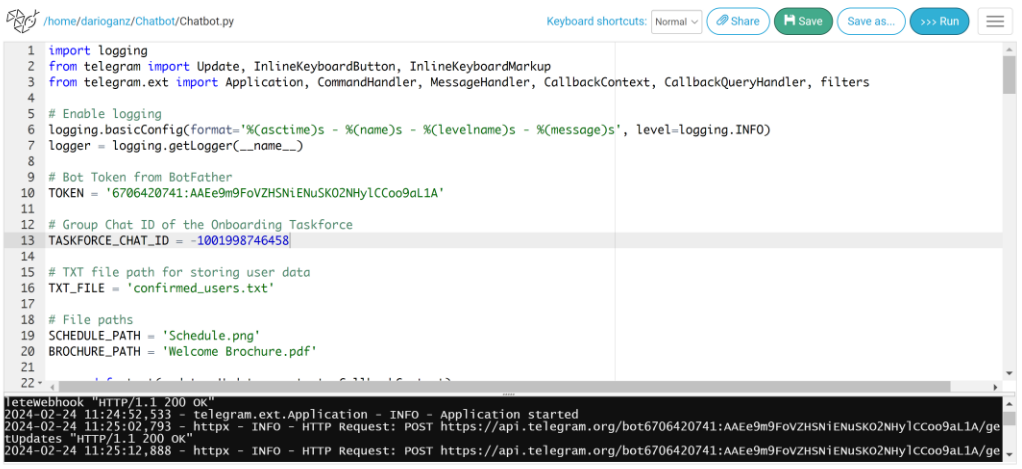


2% full – 9.7 MB of your 512.0 MB quota

Chatbot.py	2024-02-23 21:36	7.7 KB
Schedule.png	2024-02-23 20:51	55.5 KB
Welcome Brochure.pdf	2024-02-23 20:51	1.7 MB
confirmed_users.txt	2024-02-23 21:36	32 bytes

100MiB maximum size

- Navigate to your python file and press «$ open Bash console here». Type in
  «pip install python-telegram-bot»
  to download all important packages. Next run the script by pressing the blue button in the top right corner.



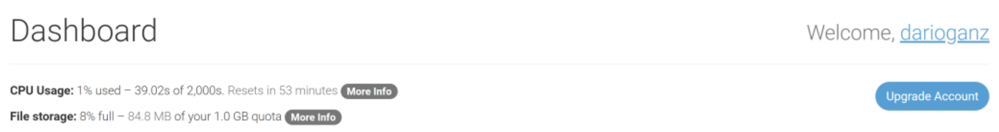
```python
import logging
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import Application, CommandHandler, MessageHandler, CallbackContext, CallbackQueryHandler, filters

# Enable logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)
logger = logging.getLogger(__name__)

# Bot Token from BotFather
TOKEN = '6706420741:AAEe9m9FoVZHSNiENuSKO2NHylCCoo9aL1A'

# Group Chat ID of the Onboarding Taskforce
TASKFORCE_CHAT_ID = -1001998746458

# TXT file path for storing user data
TXT_FILE = 'confirmed_users.txt'

# File paths
SCHEDULE_PATH = 'Schedule.png'
BROCHURE_PATH = 'Welcome Brochure.pdf'

```

```
letewebhook "HTTP/1.1 200 OK"
2024-02-24 11:24:52,533 - telegram.ext.Application - INFO - Application started
2024-02-24 11:25:02,793 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6706420741:AAEe9m9FoVZHSNiENuSKO2NHylCCoo9aL1A/ge
tUpdates "HTTP/1.1 200 OK"
2024-02-24 11:25:12,888 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6706420741:AAEe9m9FoVZHSNiENuSKO2NHylCCoo9aL1A/ge
```

- Now the bot is running! During the period before and during the onboarding week at SCL always double check each day that the bot is still running. The consoles of PythonAnywhere can run for a long time, but occasionally the servers bounce for maintenance. In our case we had to restart the bot every second day.

Always-on Tasks

If you want that the code runs 24/7 and even restarts if the servers of PythonAnywhere bounce you can start an Always-on task for 5$ per month if you press **«Upgrade Account»** dashboard:

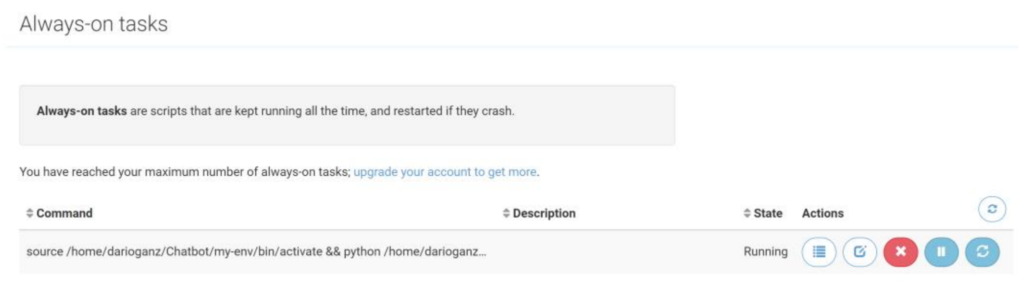

Dashboard

Welcome, darioganz

CPU Usage: 1% used – 39.02s of 2,000s. Resets in 53 minutes More Info

File storage: 8% full – 84.8 MB of your 1.0 GB quota More Info

- After a successful upgrade you need to navigate to the **«Tasks»** section:


Always-on tasks

Always-on tasks are scripts that are kept running all the time, and restarted if they crash.

You have reached your maximum number of always-on tasks; upgrade your account to get more.

Command	Description	State	Actions
source /home/darioganz/Chatbot/my-env/bin/activate && python /home/darioganz...		Running	

- To install the telegram library in the Always-on tasks section you need to create a virtual environment (venv) in the bash console to access the library (e.g. my-env is in this case the name of the venv):
  «python3.10 -m venv my-env»
- Then you have to activate the venv:
  «source /home/yourusername/something/my-env/bin/activate»
- Next on install the necessary library:
  «pip install python-telegram-bot»
- And then last but not least navigate back to the **«Tasks»** section and input this command in the Always-on tasks input field:
  «source /home/yourusername/something/my-env/bin/activate &&
  python /home/yourusername/something/script.py»
- Finally after a moment the **«State»** should switch from **«Starting»** to **«Running»** if you press the refresh button. If an error persists you can either view the task log on **«Actions»** or get help from the help page of PythonAnywhere: https://help.pythonanywhere.com/pages/AlwaysOnTasks/

8. Future Developments

Looking ahead, there are several exciting opportunities to enhance the functionality and usefulness of our Onboarding Telegram chatbot for the Smart Contracts Lab project. Here are some potential improvements and ideas for future development:

- Use the chatbot in the next onboarding week in April to enhance communication efficiency and speed
- Add more functions with buttons
- Carry out the allocation survey and the feedback survey through the chatbot