🤖 Creating a Telegram Bot in Python Using API Tokens

🧩 1. What is a Telegram Bot?

A **Telegram Bot** is an automated program that interacts with users through Telegram chat. It can perform tasks such as sending messages, retrieving information, managing groups, or integrating with external services.

Bots communicate through **Telegram's Bot API**, using **HTTP requests** or libraries that wrap those requests — most commonly **python-telegram-bot** or **Telebot (pyTelegramBotAPI)**.

⚙️ 2. How Telegram Bot Works

1. **User** sends a message to the bot on Telegram.
2. **Telegram Servers** receive the message and send it to your bot through the **Bot API**.
3. Your **Python code** processes the message and sends back a response using the same API.

🧠 3. Steps to Create a Telegram Bot

Step 1: Create a Bot via BotFather

1. Open Telegram and search for **@BotFather**.
2. Type `/start`.
3. Type `/newbot` and follow the prompts:
   - Choose a name for your bot.
   - Choose a username ending with `_bot` (e.g., `study_helper_bot`).
4. BotFather will give you a **Bot Token**, which looks like this:

   ```
   1234567890:ABCdEfGhIJKlmNoPQRstuVWxyz123456789
   ```

   ➤ **Keep this token safe!** It's what connects your Python code to your Telegram bot.

Step 2: Set Up Your Python Environment

Make sure Python is installed, then install the Telegram bot library:

```
pip install python-telegram-bot
```

(Optional alternative):

```
pip install pyTelegramBotAPI
```

Step 3: Create Your Bot Script

Using `python-telegram-bot` library:

```
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler,

# Replace with your bot token
```

```
BOT_TOKEN = "1234567890:ABCdEfGhIJKlmNoPQRstuVWxyz123456789"

# Define start command
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Hello! 👋 I'm your friendly Python bot

# Define help command
async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("You can send me a message, and I'll rep

# Define echo function (bot replies with same message)
async def echo(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = update.message.text
    await update.message.reply_text(f"You said: {text}")

# Main program
app = ApplicationBuilder().token(BOT_TOKEN).build()

# Add handlers
app.add_handler(CommandHandler("start", start))
app.add_handler(CommandHandler("help", help_command))
app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, echo))

print("Bot is running...")
app.run_polling()
```

Step 4: Run the Bot

Save the file as `telegram_bot.py` , then run:

```
python telegram_bot.py
```

Your bot will now go **online**. Open Telegram, find your bot by its username, and type `/start` .

🧩 4. How Bot Communicates

Two major methods to get messages from Telegram:

Method	Description
Polling	Your bot keeps checking Telegram servers for new messages (simpler for local testing).
Webhook	Telegram sends updates to your server's URL (better for production).

For most beginners and classroom use, **polling** is enough.

🧠 5. Adding More Features

Here are common examples to practice:

✅ Custom Commands

Add a `/time` command that returns the current time:

```
from datetime import datetime

async def time_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    await update.message.reply_text(f"⏰ Current time: {now}")

app.add_handler(CommandHandler("time", time_command))
```

✅ Reply to Specific Keywords

```
async def reply_custom(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = update.message.text.lower()
    if "hello" in text:
        await update.message.reply_text("Hi there! How can I help?")
    elif "bye" in text:
        await update.message.reply_text("Goodbye! 👋")
    else:
        await update.message.reply_text("I didn’t quite get that.")

app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, reply_custom
```

✅ Send Stickers, Images, or Files

```
async def send_photo(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await context.bot.send_photo(chat_id=update.effective_chat.id, photo="ht

app.add_handler(CommandHandler("photo", send_photo))
```

✅ Inline Keyboard Buttons

```
from telegram import InlineKeyboardButton, InlineKeyboardMarkup

async def menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
    buttons = [
        [InlineKeyboardButton("About", callback_data="about"),
         InlineKeyboardButton("Help", callback_data="help")]
    ]
    reply_markup = InlineKeyboardMarkup(buttons)
    await update.message.reply_text("Choose an option:", reply_markup=reply_

app.add_handler(CommandHandler("menu", menu))
```

🧠 6. Error Handling

You can log errors using the built-in logging module:

```
import logging

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=log
)
```

This helps you debug issues when the bot crashes or fails to reply.

☁️ 7. Deploying to the Cloud

Once you finish building and testing locally, you can host your bot on:

- **Render** (render.com)
- **Railway.app**
- **PythonAnywhere**
- **Heroku (legacy option)**

Make sure your bot runs continuously by enabling **webhooks** or background services.

🧩 8. 15 Practice Exercises on Telegram Bot Development

1. Create a bot that greets the user by name when they type `/start` .
2. Add a `/help` command listing all available commands.
3. Add a command `/reverse` that reverses the user's text.
4. Create a `/sum` command that takes two numbers and returns their sum.
5. Build a `/quiz` bot that asks a question and checks the answer.
6. Create a `/weather` bot (mocked) that returns a fake weather report.
7. Build a `/time` command showing the current system time.
8. Store user messages in a text file each time they send a message.
9. Create a simple **todo list** bot that can add and display tasks.
10. Add inline buttons to let the user choose between two options.
11. Add a command `/quote` that sends a random motivational quote.
12. Create an admin-only `/broadcast` command.
13. Make the bot send a random photo using a public API.
14. Handle invalid inputs gracefully with try-except.
15. Deploy your bot on **Render** and test live.