```
import numpy as np
import time
import os
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch


# checkpoint
checkpoint = "microsoft/DialoGPT-medium"
# download and cache tokenizer
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
# download and cache pre-trained model
model = AutoModelForCausalLM.from_pretrained(checkpoint)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to acces
  warnings.warn(
tokenizer_config.json: 100%  614/614 [00:00<00:00, 4.92kB/s]
vocab.json: 100%  1.04M/1.04M [00:00<00:00, 6.64MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 7.57MB/s]
config.json: 100%  642/642 [00:00<00:00, 10.1kB/s]
pytorch_model.bin: 100%  863M/863M [00:10<00:00, 61.5MB/s]
generation_config.json: 100%  124/124 [00:00<00:00, 2.11kB/s]
```

```
# Build a ChatBot class with all necessary modules to make a complete conversation
class ChatBot():
    # initialize
    def __init__(self):
        # once chat starts, the history will be stored for chat continuity
        self.chat_history_ids = None
        # make input ids global to use them anywhere within the object
        self.bot_input_ids = None
        # a flag to check whether to end the conversation
        self.end_chat = False
        # greet while starting
        self.welcome()

    def welcome(self):
        print("Initializing ChatBot ...")
        # some time to get user ready
        time.sleep(2)
        print('Type "bye" or "quit" or "exit" to end chat \n')
        # give time to read what has been printed
        time.sleep(3)
        # Greet and introduce
        greeting = np.random.choice([
            "Welcome, I am ChatBot, here for your kind service",
            "Hey, Great day! I am your virtual assistant",
            "Hello, it's my pleasure meeting you",
            "Hi, I am a ChatBot. Let's chat!"
        ])
        print("ChatBot >>  " + greeting)

    def user_input(self):
        # receive input from user
        text = input("User    >> ")
        # end conversation if user wishes so
        if text.lower().strip() in ['bye', 'quit', 'exit']:
            # turn flag on
            self.end_chat=True
            # a closing comment
            print('ChatBot >>  See you soon! Bye!')
            time.sleep(1)
            print('\nQuitting ChatBot ...')
        else:
            # continue chat, preprocess input text
            # encode the new user input, add the eos_token and return a tensor in Pytorch
            self.new_user_input_ids = tokenizer.encode(text + tokenizer.eos_token, \
                                                       return_tensors='pt')

    def bot_response(self):
        # append the new user input tokens to the chat history
        # if chat has already begun
        if self.chat_history_ids is not None:
            self.bot_input_ids = torch.cat([self.chat_history_ids, self.new_user_input_ids], dim=-1
        else:
            # if first entry, initialize bot_input_ids
            self.bot_input_ids = self.new_user_input_ids

        # define the new chat_history_ids based on the preceding chats
        # generated a response while limiting the total chat history to 1000 tokens,
        self.chat_history_ids = model.generate(self.bot_input_ids, max_length=1000, \
                                               pad_token_id=tokenizer.eos_token_id)

        # last ouput tokens from bot
        response = tokenizer.decode(self.chat_history_ids[:, self.bot_input_ids.shape[-1]:][0], \
                                    skip_special_tokens=True)
        # in case, bot fails to answer
        if response == "":
            response = self.random_response()
        # print bot response
        print('ChatBot >>  '+ response)

    # in case there is no response from model
    def random_response(self):
        i = -1
        response = tokenizer.decode(self.chat_history_ids[:, self.bot_input_ids.shape[i]:][0], \
```

```
                                skip_special_tokens=True)

# build a ChatBot object
bot = ChatBot()
# start chatting
while True:
    # receive user input
    bot.user_input()
    # check whether to end chat
    if bot.end_chat:
        break
    # output bot response
    bot.bot_response()
```

```
Initializing ChatBot ...
Type "bye" or "quit" or "exit" to end chat

ChatBot >>  Hi, I am a ChatBot. Let's chat!
User    >> hello
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  Hello! :D
User    >> can you do me a favour
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  Sure! :D
User    >> suggest me a good song
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  I don't know, I'm not a big fan of songs.
User    >> what else did you like?
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  I don't really listen to music.
User    >> movies??
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  I don't really watch movies.
User    >> ok...
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  I don't really watch movies either.
User    >> ok bye
A decoder-only architecture is being used, but right-padding was detected! For correct generation results, please se
ChatBot >>  bye bye
User    >> exit
ChatBot >>  See you soon! Bye!

Quitting ChatBot ...
```