| 자유 게시판 | >

[스압] 디코 코인 탑승해서 음성인식 및 목소리가 있는 비서 만들기

윤 챗봇 입문자 1:1 채팅
2020.06.21. 20:19 조회 353

안녕하세요. 그동안 뭘 쓸까, 이걸 쓰기엔 시간이 너무 아까운데 싶다가도
예전에 한 번 시도해본 디스코드 봇이 요즘 유행하길래 저도 코인 탑승을 해봤습니다.

그런데, 남들이 하는 거 다 하면 제가 뭣하러 글을 쓸까요? 다른 분들이 더 잘 할텐데.
이번 글에서는 봇이 음성 채널에 참가하고 실시간으로 음성을 분석, 음성으로 대답을 해주는 음성 인식 봇을 만드려고 합니다.

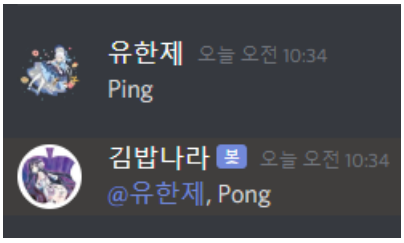
- □ 음성채널에 참가해서 **mp3** 재생
- □ 실시간 음성 명령어 인식
  - Speech To Text
  - 자연어 분석
- □ 유한 오토마톤으로 상태 관리
- □ 명령어 답변 음성으로 재생

□ 음성채널에 참가해서 **mp3** 재생

디스코드가 지원하는 API 를 사용할 수 있는 언어는 다양하지만, 제게 있어서 가장 개발속도도 빠르고 능숙하게 다룰 수 있는 자바스크립트를 지원하는, Discord.JS 를 사용하도록 하겠습니다. 아, 요즘 타입스크립트 연습도 하고 있으니 이왕 타입스크립트로 개발하도록 하죠.

타입스크립트의 d.ts 파일은, 여기에서 다운받아 사용했습니다.

뭐, 일단 디스코드 앱을 설정해서 봇 토큰을 발급하고, "Ping" 이라 보내면 "Pong" 하고 돌아오는 과정까진 모두 쉽게 하시리라고 생각합니다. 아니라면 다른 분들의 훌륭한 강좌를 보고 오세요. 저는 강좌가 아니라 그냥 놀이니까.



의외로 음성채널 채팅에 참가하는 것은 매우 간단하더라고요.
코드 한 줄 이면 참가가 가능했습니다.

보이스 채널에 참가한 상태로, 메시지를 보내 트리거를 추가하면 참가가 가능했습니다.

```
import Discord from 'discord.js';
import { Message } from 'discord.js';


const client = new Discord.Client();

client.on('message', async (msg: Message) => {
    if ( !msg.guild ) return;

    if ( msg.content === "!참가" ) {
        const connection = await msg?.member?.voice?.channel?.join();
    }
});
```

Colored by Color Scripter

위 코드의 결과

이제, mp3 아무거나 다운을 받고 재생을 시켜봅시다.
mp3 등 오디오파일을 다루려면 ffmpeg 가 필요하다고 합니다.
설치해줍니다.

```
npm install ffmpeg-static
```

```
import Discord from 'discord.js';
import { Message, VoiceConnection } from 'discord.js';
import path from 'path';


const client = new Discord.Client();
var connection: VoiceConnection;

client.on('message', async (msg: Message) => {
    if ( !msg.guild ) return;

    if ( msg.content === "!참가" ) {
        connection = await msg?.member?.voice?.channel?.join() as VoiceConnection;
    }

    if ( msg.content === "!재생" ) {
        const dispatcher = connection.play(path.join(__dirname, 'assets', 'music.mp3'));
    }
});

```

잘 재생이 되는 것을 확인했습니다.

□ 실시간 음성 명령어 인식

Speech To Text (STT) 라고 들어보신 분들이 계실겁니다. 음성 파일에서의 말을 텍스트로 변환시켜주는 기술이죠.
보통 이것은, 길이가 한정된 음성파일의 내용을 분석하는 것입니다. 하지만 우리는 실시간으로, 언제 말을 할지 언제까지 말을 할지 모르기 때문에 조금 어려움이 있을 수 있습니다.

그런데, Discord.JS 에서 훌륭한 기능을 제공하고 있었습니다.
디스코드에 그게 있잖아요? 설정에 보면 녹음모드 음성감지 말이죠. 이게 봇에도 있었습니다.
우리는 목소리가 mp3 로 변형되어 나오기를 기대합니다. 하지만 디스코드는 pcm 과 opus 포맷만 지원했습니다.

사실 이후 사용할 STT 엔진이 pcm 도 지원했었으나, mp3 로 변환하여 분석하는 것이 훨씬 뛰어난 수준의 정확도를 보여주었습니다. 그렇기 때문에 Lame 라는 모듈을 사용하려고 했습니다.

리눅스 우분투 (데비안) 기준 다음 명령어를 사용합니다.

```
sudo apt install lame
npm install node-lame
```

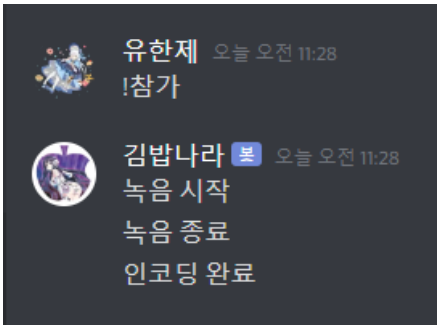
그리고 다음 코드를 사용해서

```
import Discord from 'discord.js';
import { Message, VoiceConnection, VoiceReceiver, User } from 'discord.js';
import path from 'path';
import fs from 'fs';
import { Lame } from 'node-lame';
import { Readable } from 'stream';

const SILENCE_FRAME = Buffer.from([0xF8, 0xFF, 0xFE]);
class Slience extends Readable {
    _read() {
```

```
        this.push(SILENCE_FRAME);
        this.destroy();
    }
}


const client = new Discord.Client();
var connection: VoiceConnection;
var receiver: VoiceReceiver;
var isRecord: boolean = false;

client.on('message', async (msg: Message) => {
    if ( !msg.guild ) return;

    if ( msg.content === "!참가" ) {
        connection = await msg?.member?.voice?.channel?.join() as VoiceConnection;
        receiver = connection?.receiver as VoiceReceiver;

        connection.play(new Slience(), { type: 'opus' });
        connection.on('speaking', async (user: User, speaking: boolean) => {
            if ( speaking && !isRecord ) {
                isRecord = true;
                msg?.channel?.send('녹음 시작');

                const date = new Date().getTime();
                const audioStream = receiver.createStream(user, { mode: 'pcm' });

                const pcmBufferChunks: Buffer[] = [];
                audioStream.on('data', (d: Buffer) => {
                    pcmBufferChunks.push(d);
                });
                audioStream.on('end', () => {
                    msg.channel.send('녹음 종료');
                    isRecord = false;
                    const pcmBuffer = Buffer.concat(pcmBufferChunks);

                    const encoder = new Lame({
                        output: './audio-files/' + date + '.mp3',
                        bitrate: 48,
                    }).setBuffer(pcmBuffer);
                    encoder.encode()
                        .then(() => {
                            msg.channel.send('인코딩 완료');
                        })
                });
            }
        });
    } else if ( msg.content === "!해제") {
        await msg?.member?.voice?.channel?.leave();
    }
});

```

Colored by Color Scripter

그래서 디스코드에 참가시키고 말을 하면 녹음한 내용을 mp3 파일로 저장합니다!



목소리가 좋진 않아서 올리기 좀 그렇지만, 실제 녹음됐다는 것을 증명하기 위해 업로드했습니다.

Speech To Text

이제 서론이 끝나고, 실제로 해봐야 할 Speech To Text 를 사용해 봅시다.
우리는 언제나 그랬듯 대형 회사인 구글의 기술을 빌려쓸 것입니다.
구글 speech 는 REST API 를 지원합니다. 자세한 프로젝트 설정은, speech 빠른시작 문서와 speech v1 beta1 api 문서를 보고 제가 이전에 썼었던 자연어 파싱하고 감정분석 하기를 참고하시기 바랍니다.

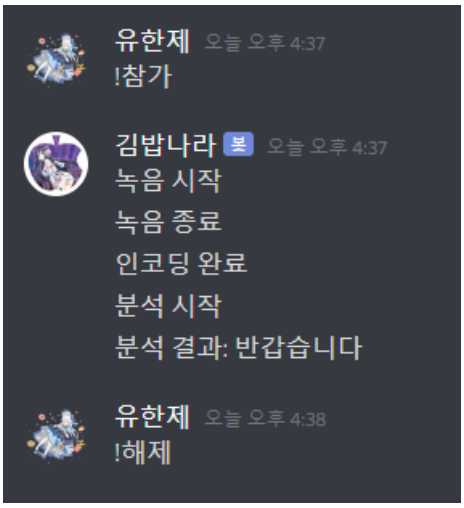
위에선 mp3 를 파일로 뽑아냈지만, 그것은 직접 확인을 하기 위한 절차일 뿐, 우리는 다시 버퍼로 데이터를 전송할 것이기 때문에 파일로 뽑아내는 것이 아닌 버퍼로 뽑아내는 과정을 거쳤습니다.

```
import Discord from 'discord.js';
import { Message, VoiceConnection, VoiceReceiver, User } from 'discord.js';
import path from 'path';
import fs from 'fs';
import { Lame } from 'node-lame';
import { Readable } from 'stream';
import axios from 'axios';
import { AxiosRequestConfig } from 'axios';

// API Key 를 global 객체에 선언하기 위한 타입 재정의.
declare global {
    namespace NodeJS {
        interface Global {
            apiKey: string;
        }
```

```
    }
}

const SILENCE_FRAME = Buffer.from([0xF8, 0xFF, 0xFE]);
class Slience extends Readable {
    _read() {
        this.push(SILENCE_FRAME);
        this.destroy();
    }
}


const client = new Discord.Client();
var connection: VoiceConnection;
var receiver: VoiceReceiver;
var isRecord: boolean = false;

client.on('message', async (msg: Message) => {
    if ( !msg.guild ) return;

    if ( msg.content === "!참가" ) {
        connection = await msg?.member?.voice?.channel?.join() as VoiceConnection;
        receiver = connection?.receiver as VoiceReceiver;

        connection.play(new Slience(), { type: 'opus' });
        connection.on('speaking', async (user: User, speaking: boolean) => {
            if ( speaking && !isRecord ) {
                isRecord = true;
                msg?.channel?.send('녹음 시작');

                const date = new Date().getTime();
                const audioStream = receiver.createStream(user, { mode: 'pcm' });

                const pcmBufferChunks: Buffer[] = [];
                audioStream.on('data', (d: Buffer) => {
                    pcmBufferChunks.push(d);
                });
                audioStream.on('end', async () => {
                    msg.channel.send('녹음 종료');
                    isRecord = false;
                    const pcmBuffer: Buffer = Buffer.concat(pcmBufferChunks);

                    const encoder = new Lame({
                        output: 'buffer',
                        bitrate: 48,
                    }).setBuffer(pcmBuffer);

                    await encoder.encode();
                    msg.channel.send('인코딩 완료');

                    const mp3Buffer = encoder.getBuffer();

                    const audio = {
                        content: mp3Buffer.toString('base64'),
                    };
                    const config = {
                        encoding: 'MP3',
                        sampleRateHertz: 48000,
                        languageCode: 'ko-KR',
                    };

                    const request: AxiosRequestConfig = {
                        url: 'https://speech.googleapis.com/v1p1beta1/speech:recognize',
                        method: 'post',
                        params: {
                            alt: 'json',
                            key: global.apiKey,
                        },
                        data: {
```

```
                            config,
                            audio,
                        }
                    };

                    msg.channel.send("분석 시작");
                    try {
                        const res = await axios(request);
                        const { results } = res.data;
                        const scriptArr: string[] = [];

                        if ( results ) {
                            for ( const result of results ) {
                                scriptArr.push(result.alternatives[0].transcript);
                            }
                        }

                        const transcription = scriptArr.join('\n');

                        msg.channel.send('분석 결과: ' + transcription);
                    } catch(err) {
                        console.error(err);
                    }
                });
            }
        });
    } else if ( msg.content === "!해제") {
        await msg?.member?.voice?.channel?.leave();
    }
});
```

Colored by Color Scripter



자연어 분석

사실, 자연어 분석에 대한 내용은 이미 저번에 진행했었습니다. 이번에도 별 다를 것이 없습니다.
실제 NLP 는 더 이후에 다룰 예정이니, 제가 이번에 사용할 내용을 자세히 보고 싶으시다면 아래 게시글을 참고해 보세요.

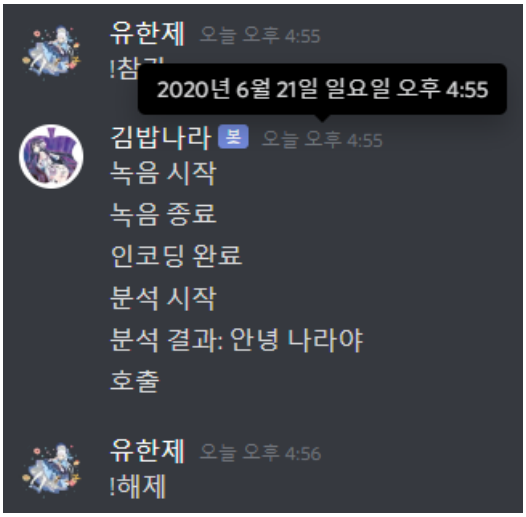
https://cafe.naver.com/nameyee/19295

그데, 제가 봇을 호출할 때는 같을 것이 없기 때문에 위 모듈을 사용하는 것이 아니

근데, 제가 봇을 오랫동안 때는 거를 것이 없기 때문에 위 모듈을 사용하는 것이 아닌
단순한 정규식 처리를 하도록 하겠습니다.

```
if ( transcription.match(/안녕\s*나라야/) ) {
    msg.channel.send("호출");
}
```
Colored by Color Scripter

아까 코드에 위 세줄 추가한 것이 끝입니다.



좀 부족한 것 같아도 잘 동작합니다.
시동어는 "안녕 나라야" 입니다.
하이 빅스비, 시리야, 오케이 구글 같은 것이죠.

□ 유한 오토마톤으로 상태 관리

유한 오토마톤 또는 유한 상태 기계라고 불리는 FSM(Finite State Machine) 기술은, 프로그램이 특정 유일한 상태값을 가진다 가정하고 각 상태별 동작과 처리에 대한 내용을 설계하는 것입니다.

이는 지금도 사용되는지 모르겠지만, 한 때 게임의 A.I 에 자주 사용되는 기술이었습니다.
원래는 제가 사용하던 방식대로 하려 했으나, C 언어가 아닌 것을 알고 EventEmiit 를 사용하도록 하겠습니다.

```
import { EventEmitter } from 'events';

export enum FSM_STATUS {
    NONE      = 0,
    PENDING   = 0x1,
    CALLED    = 0x2,
    COMMAND   = 0x4,
}

const psleep = async (msec: number) => {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, msec);
    });
}

```

```
export class FSM extends EventEmitter {
    public before: FSM_STATUS = FSM_STATUS.NONE;
    public now: FSM_STATUS = FSM_STATUS.NONE;

    constructor() {
        super();
        this.before = FSM_STATUS.PENDING;
    }

    public change(status: FSM_STATUS, ...args) {
        this.before = this.now;
        this.now = status;
        this.emit('status-' + (status | this.before), ...args);
    }

    public status(now: FSM_STATUS, before: FSM_STATUS, callback: Function) {
        this.on('status-' + (now | before), callback as any);
    }
}

const fsm = new FSM();
global.fsm = fsm;
fsm.status(FSM_STATUS.CALLED, FSM_STATUS.PENDING, async () => {
    console.log("CALLED");
});
fsm.status(FSM_STATUS.COMMAND, FSM_STATUS.CALLED, async (res) => {
    fsm.change(FSM_STATUS.PENDING);
    console.log("COMMAND", res.answer, res.cmd);
});
```

Colored by Color Scripter

이런식으로 FSM 상태를 관리하는 파일을 따로 만들었습니다.
그럼, 저것들은 이제 누가 호출할 것이냐? 당연히 녹음된 음석을 분석한 결과를 가지고 호출을 해야겠죠.

```
...

msg.channel.send('분석 결과: ' + transcription);

if ( transcription.trim() ) {
    if ( transcription.match(/안녕\s*나라야/) ) {
        global.fsm.change(FSM_STATUS.CALLED);
    } else if ( global.fsm.now === FSM_STATUS.CALLED ) {
        const res = Indent(transcription);
        global.fsm.change(FSM_STATUS.COMMAND, res);
    }
}

...
```

Colored by Color Scripter

위와 같이 말입니다. 그리고, 더 위에 보면 fsm.status 를 CALLED -> COMMAND 로 갔을 때만 반응하게 해뒀습니다.
그러니 먼저 호출되지 않으면 어떤 말을 해도 무시하게 될것입니다. 자연어 처리는 다음과 같습니다.

```
...

const indentList: Indent[] = [
    {
        "key": "music",
        "pos": "subject",
        "indent": ["노래", "음악", "뮤직"],
        "sub-indent": [
            {
                "key": "play",
                "pos": "doit",
                "indent": [ "틀어\\S*", "재생\\S*" ],
                "finish": true,
```

```
            },
        ],
    },
];


...


const answerList: Answer[] = [
    {
        "key": "music",
        "sub-answer": [
            {
                "key": "play",
                "answer": "네. 노래를 재생해 드릴게요",
                "cmd": "music.play",
            },
        ],
    }, // save
];


...

export default (txt: string) => {
    const deep = searchIndent(txt);
    const answer = searchAnswer(deep);

    return answer;
}
```

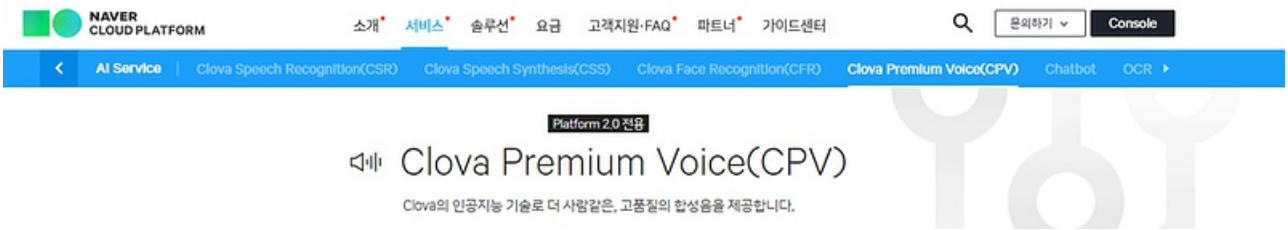
Colored by Color Scripter

그리고 이전엔, Indent 함수를 실행하게 되면 무조건 답변 문장만 나왔었는데, cmd 와 답변을 같이 주도록 하였습니다.

□ 명령어 답변 음성으로 재생

빅스비! 시리! 인공지능 비서!
다들 예쁜 목소리를 가지고 있습니다. 그러니 우리의 김밥나라 비서님도 예쁜 목소리를 가져야 합니다.
보통의 TTS 엔진으로는 그렇게 예쁜 목소리가 나오질 않거든요.

그래서 준비한 것이, 네이버 클라우드의 프리미엄 보이스입니다.



제가 여러 TTS를 써본 결과, 이 목소리가 가장 자연스럽고 예뻤습니다. 심지어 파파고의 규리보다 더 예쁜 목소리가 나왔습니다.

하지만 이것은 유료입니다. 때문에 매번 새로 호출해버리면 요금이 어마어마하게 나올 것 같습니다.
그러니 한 번 요청했던 TTS 음성은, 저장해두고 재사용하는 것이 이득인 것 같습니다.

```
// native modules
import fs from 'fs';
import path from 'path';

// package modules
import httpReq from 'request';

const getPath = (p: string) => path.join(process.cwd(), p);

const tts = require(getPath('./tts.json'));

const ttsRequest = (txt: string) => {
    return new Promise((resolve, reject) => {
        const options = {
            url: 'https://naveropenapi.apigw.ntruss.com/voice-premium/v1/tts',
            method: 'post',
            form: {
                speaker: "nara",
                speed: 0,
                text: txt,
            },
            headers: {
                'X-NCP-APIGW-API-KEY-ID': global.naraId,
                'X-NCP-APIGW-API-KEY': global.naraKey,
            },
        };

        const date = Date.now();
        const fname = getPath('audio-files/' + date + '.mp3');
        const writeable = fs.createWriteStream(fname);

        try {
            const req = httpReq(options, (err) => {
                if ( err ) {
                    reject(err);
                }
            });
            req.pipe(writeable);
        } catch(err) {
            reject(err);
        }

        writeable.on('close', () => {
            resolve(fname);
        });
    });
}

```

```
const say = (fname: string) => {
    return new Promise((resolve, reject) => {
        try {
            const dispatcher = global.connection.play(fname);
            dispatcher.on('finish', () => {
                resolve();
            });
        } catch(err) {
            reject(err);
        }
    });
}

export default async (txt: string) => {
    let fname = "";
    if ( tts[txt] ) {
        fname = tts[txt];
    } else {
        // 새로운 요청
        fname = await ttsRequest(txt) as string;
        tts[txt] = fname;
        fs.writeFile(getPath('./tts.json'), JSON.stringify(tts, null, '\t'), { encoding: 'utf8'
    }

    return say(fname);
}

```

axios 모듈을 놔두고 지금은 사용되지 않는 request 모듈을 사용한 이유는, 이전 작업을 할 때 axios 모듈을 사용해서 해결해 보려고 했었던 경험이 있거니와 다른 프로젝트 코드의 재사용 때문입니다.

어쨌든, tts.json 에 음성 파일이 있으면 해당 파일을 재생하고, 없으면 새로운 tts 를 요청해 파일을 생성 후 재생합니다.
최종적인 FSM 은 다음과 같습니다.

```
...

const fsm = new FSM();
global.fsm = fsm;
fsm.status(FSM_STATUS.CALLED, FSM_STATUS.PENDING, async () => {
    Read('부르셨나요?');
    console.log("CALLED");
});
fsm.status(FSM_STATUS.COMMAND, FSM_STATUS.CALLED, async (res) => {
    fsm.change(FSM_STATUS.PENDING);

    await Read(res.answer);
    switch ( res.cmd ) {
        case "music.play":
            global.connection.play(
                path.join(process.cwd(), 'music.mp3')
            );
            break;
    }
});
```


그럼, 제대로 동작하는지 확인해 보아야겠죠.

정상 동작하는 것을 확인했습니다!

이번 프로젝트는 깃허브에 올려두었습니다. 전체 소스가 궁금하신 분들은 참고하세요.

https://github.com/Tree-Some/discord-speech-bot

이상, 인공지능 비서를 만들어 본 챗봇 입문자 윤이였습니다.

읽어주셔서 감사합니다.

윤님의 게시글 더보기 >

좋아요 7 댓글 18 공유 | 신고

댓글 등록순 최신순

댓글알림

윤 작성자
장장 2일이 걸린 글입니다.
왜냐고요? 중간 중간 게임하느라고요.
2020.06.21. 20:19 답글쓰기

알랑뿡까
사랑해요
2020.06.21. 20:22 답글쓰기

윤 작성자
예쁘신가요
2020.06.21. 20:24 답글쓰기

BennyK
입문자요?
2020.06.21. 20:38 답글쓰기

윤 작성자
네 등급이 그렇습니다
2020.06.21. 20:40 답글쓰기

BennyK
윤 저도 입문할래요
2020.06.21. 20:49 답글쓰기

리페
와 대단해요 ㄷㄷ
내용은 이해가안가지만 결과물을 보니 입이 떡 벌어지네요
2020.06.21. 20:38 답글쓰기

윤 작성자
감사합니다

2020.06.21. 20:40 답글쓰기

성빈
와! 윤님의 목소리!
2020.06.22. 07:13 답글쓰기

윤 작성자
부끄럽습니다
2020.06.22. 07:41 답글쓰기

윤 위해 웃어줘
d.ts!
2020.06.22. 07:36 답글쓰기

윤 작성자
d.ts가 없으면 타입을 알 수가 없죠
2020.06.22. 07:41 답글쓰기

윤 위해 웃어줘
윤 ㅋㅋ 역시 배우신분
2020.06.22. 07:46 답글쓰기

윤 위해 웃어줘
윤 위해 웃어줘 전 리액트 강의 준비중입니다(?)

2020.06.22. 07:47 답글쓰기

윤 작성자

윤 위해 웃어줘 봇 커뮤에서 리액트 강의가 나오다니 ㄷㄷ
2020.06.22. 07:47 답글쓰기

윤 위해 웃어줘
윤 낄
2020.06.22. 07:48 답글쓰기

야옹
윤 위해 웃어줘 (먼지 모르는 1인)
2020.06.25. 09:54 답글쓰기

윤 위해 웃어줘
야옹 삐빅 정상입니다
2020.06.25. 13:12 답글쓰기

Hibot
댓글을 남겨보세요
등록

글쓰기 답글 목록 ▲ TOP

'| 자유 게시판 |' 게시판 글

이 게시판 새글 구독하기

카페 파싱이 데이터 많이 먹네요 [1]	공돌이	2020.06.22.
컴파일풀림현상 [4]	잠깐만요이	2020.06.22.
[스압] 디코 코인 탑승해서 음성인식 및 목소리가 있는 비서 만들기 [18]	윤	2020.06.21.
4000 ㅊㅊ.. [1]	처럼123	2020.06.21.
4000명 [2]	SP청정	2020.06.21.

1 2 3 전체보기

이 카페 인기글

이사람은 나오세여
폰수리가
♡0 💬19


간단 계산기 입니다
deadmau5
♡0 💬7

이카페는 팩트를 쓰면 날아가는군요.
tomohong
♡2 💬9

옵젯참여 하고싶지만 이용자보호조치 때문에 참여가 제한...
이발
천국봇
카알봇
0 13
0 5

계산기 소스
JSR
0 15

카톡봇 사람들은 꼭봐야됨

메신저R]카톡방 들어와 채팅치면 환영해주는 소스
자동읽음소스
deadmau5
tomahong
0 15
0 9