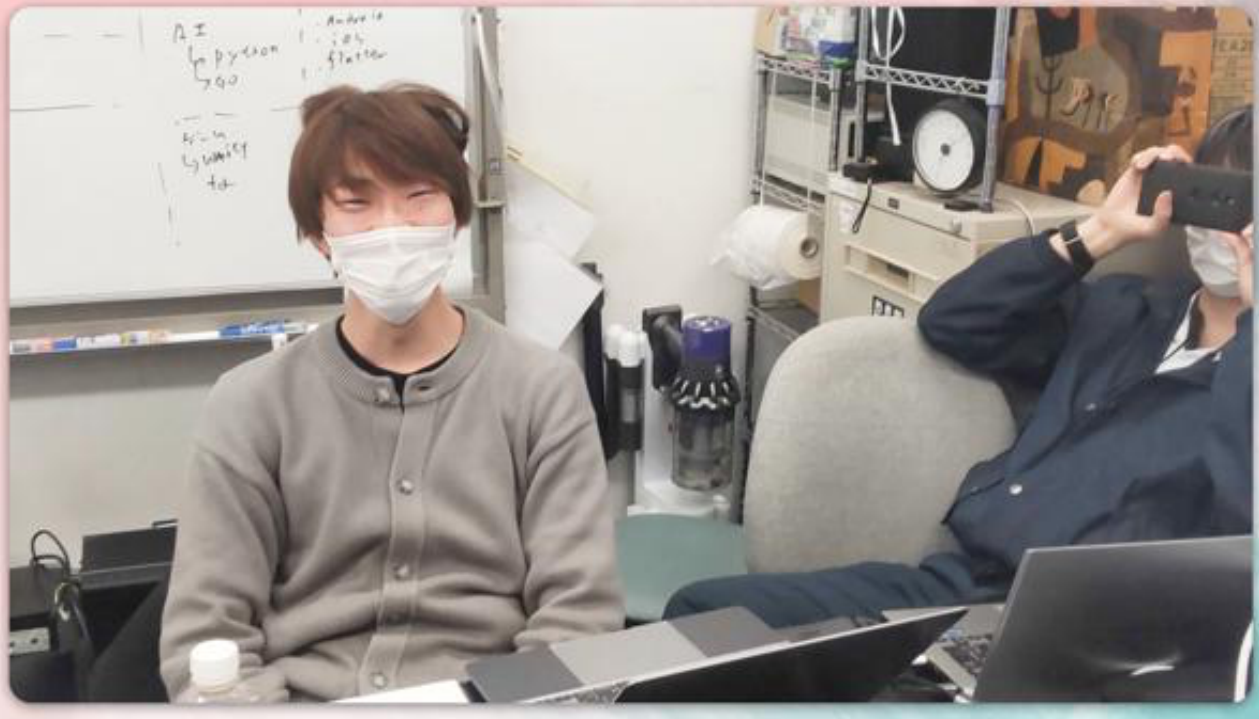
シス研の技術本
テスト作成
表紙



目次

第 1 章　これは chapter　2
1.1　これは section　2

第 2 章　DiscordBot を作ってみよう!　4
2.1　DiscordBot を作ってみよう　4
2.2　実行環境・使用技術　5
2.3　ローカル環境で Bot が動作するようにする　5
2.4　まとめ　8

第 3 章　リポジトリ作成後に設定しておきたいこと　9
3.1　はじめに　9
3.2　ファイルの設定　10
3.3　インフラ回り　13
3.4　おわりに　14

第 4 章　これは chapter　15
4.1　これは section　15

第 5 章　これは chapter　17
5.1　これは section　17

第 6 章　これは chapter　19
6.1　これは section　19

第 7 章　これは chapter　21
7.1　これは section　21

第 8 章　これは chapter　23
8.1　これは section　23

1 これは chapter

1.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

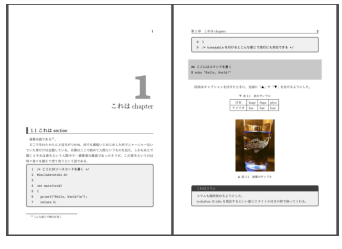
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 1.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 1.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

2

DiscordBot を作ってみよう!

2.1 DiscordBot を作ってみよう

2.1.1 はじめに

はじめまして、suda です。私は Discord で人とチャットをしている時に同じ会話が頻繁に続き、これ Bot で返事をするようにしたら返事をする手間が省けるし面白いのでは？と思い Bot を作ることにしました。発想がひどいって！？まあでも自分の発想したものを形にすることが面白いことだと思うので今回はそこには目を瞑りましょう…もちろん自分が送ったメッセージに対して Bot に返答させることもできるので、自分だけのオリジナル DiscordBot を作ってみましょう！

2.1.2 何を作るのか

Discord のサーバで特定のメッセージが来たら、特定のメッセージを返す Discord の Bot を作ります。サンプルプログラムを参照したい方は以下の URL からご覧下さ

い。[*1] 例えば自分が「仕事終わった」と言うと Bot が「お疲れ様」と返してくれます。

2.2 実行環境・使用技術

- Python 3.10.8

2.3 ローカル環境で Bot が動作するようにする

まずはローカル環境で Bot が動作するようにしてみます。

2.3.1 Bot の作成・管理をする

初めに、機能などはまだついていない Bot を Discord のポータルサイトから作成します。Discord の Bot の作り方 (メモ) という記事の「1.Discord 上の Bot の作成」を見ながら Bot を作成してみて下さい。[*2]

2.3.2 ファイルの作成

Bot を実行する Python ファイルを作ります。

```
mkdir message_discord_bot
cd message_discord_bot
touch main.py
```

[*1] 今回作る DiscordBot のサンプルプログラム https://github.com/sudamichiyo/Discord_Bot_sampleprogram

[*2] Discord の Bot の作り方 (メモ)https://note.com/exteoi/n/nf1c37cb26c41(参照 2023.3.29)

2.3.3 discord.py の準備

ここからは discord.py のドキュメントを見ながら環境構築をしていきます。[*3] Python で Discord の API を操作するために必要なライブラリをインストールします。先ほど作成したディレクトリにアクセスして、以下のコマンドで discord.py をインストールします。

```
python -m pip install -U discord.py
```

次に、先ほど作成した main.py を以下のソースコードに書き換えます。

```
1 import discord
2
3 class MyClient(discord.Client):
4     async def on_ready(self):
5         print(f'Logged on as {self.user}!')
6
7     async def on_message(self, message):
8         print(f'Message from {message.author}
9                                         : {message.content}')
10
11 intents = discord.Intents.default()
12 intents.message_content = True
13
14 client = MyClient(intents=intents)
15 client.run('my token goes here')
```

ここで、以下のボットに関する 2 つの設定を Discord のポータルサイトから設定してください。

[*3] discord.py ドキュメント https://discordpy.readthedocs.io/ja/latest/intro.html#basic-concepts(参照 2023.3.29)

- ポータルサイトの「Bot」からトークンを取得する
- ポータルサイトの「Bot」の「MESSAGE CONTENT INTENT」を有効にする

'my token goes here' は取得した Bot のアクセストークンを書きます。以上の設定が終わったところで python3 main.py を実行すると、Bot のサーバが立ち上がります。Bot サーバ起動後に Bot のいるサーバでメッセージを投げると、コマンドライン上に「書いた人」と「メッセージ」がそのまま出力されます。

2.3.4 環境変数の設定

ソースコードに直接トークンを書いてしまうと、Github でソースコードをホスティングするときにトークンキーが他の人にバレてしまいます。これを防ぐために.env ファイルを作成して、その中に Discord のアクセストークンを書きます（下記参照）。

```
1 DISCORD_TOKEN='My token goes here'
```

Python の中で.env ファイルに書かれている変数を取得するために dotenv というライブラリを使用します。以下のようにインストールします。

```
pip install python-dotenv
```

インストール後に main.py に下記のコードを付け加えて下さい。
main.py の import discord と class MyClient の間に以下のコードを追加します。

```
1 import os
2 from dotenv import load_dotenv
3 load_dotenv()
```

そして、最後の行を以下のように書き換えて下さい。

```
1 client.run(os.environ['DISCORD_TOKEN'])
```

書き換えたあとに python3 main.py を実行すると，先ほどと同じようにメッセージの受け取りをしてくれるサーバーサイドアプリケーションが立ち上がります。

2.3.5 Bot が特定のワードに反応して、特定のメッセージを返答する機能をつける

プログラムを起動して正常にサーバーサイドアプリケーションがメッセージを受け取れるようになったら、Bot が特定のワードに反応して、特定のメッセージを返答する機能をつけていきます。Bot に機能をつけるには上記のソースコードの 8 行目と 10 行目の間に以下のコードを付け足していきます。

```
1 # メッセージを書いた人が Bot なら処理終了
2 if message.author.bot:
3     return
4 channel = message.channel
5 if message.content == '仕事終わった':
6     await channel.send('お疲れ様')
```

付け足したコードの解説をしていきます。
2,3 行目でメッセージを書いた人が Bot なら処理を終了させています。
4 行目でメッセージが投稿されたチャンネル取得しています。
5 行目の message.content はメッセージの内容で、今回の場合「仕事終わった」というメッセージをチャンネルに投稿すると、メッセージが投稿されたチャンネルに Bot が「お疲れ様」と返答します。

2.4 まとめ

今回は Discord の Bot の作り方を説明しました。上記の「仕事終わった」や「お疲れ様」に当たる部分を変えたりして自分好みに改良してみて下さい。

3

リポジトリ作成後に設定しておきたいこと

3.1 はじめに

こんにちは。hihumikan です。

本チャプターでは「リポジトリ作成後に設定しておきたいこと」をご紹介します。私自身が次回プロジェクト開始する際に、こういった設定や技術を使うだろうというものをまとめました。

ただし、これら全ての内容をプロジェクトに適用出来るものではないため、ご利用の環境や用途に合わせて利用していただければと思います。

3.1.1 対象読者

対象読者は、Git/GitHub を利用した開発を行う初学者の方を想定しています。

3.2 ファイルの設定

リポジトリ作成後に設定しておきたい「ファイルの設定」についてご紹介します。

3.2.1 .gitignore

.gitignore は、Git による管理から除外したいファイルやディレクトリを指定するための設定ファイルです。

例えば、MacOS の場合、ディレクトリ毎に.DS_Store というファイルが自動的に生成されます。このファイルは、ディレクトリの meta 情報を記録しますが、通常の開発において共有する必要がないため、Git の管理対象外としておくことが望ましいです。

また、.env などの環境変数を利用してプログラムを動かす場合、.env ファイルには、パスワードや秘密鍵などの情報が含まれていることが多いため、これも Git の管理対象外としておくことが望ましいです。

リスト 1.1 のようなテキストファイルを Git の管理下に置くだけで、Git から管理対象外として扱うことができます。

リスト 1.1 .gitignore

```
1 .DS_Store
2 .env
```

リポジトリを共有する場合、.gitignore ファイルを共有することで、開発メンバー全員が同じ設定を利用できるため、必要なファイルだけが Git の管理下に置かれるようになります。

3.2.2 .gitattributes

.gitattributes は、特定のファイルに対して Git の挙動を変更するための設定ファイルです。主に、改行コードやファイルの文字コードを指定するために利用されます。

改行コードを指定する理由としては、Windows と MacOS では改行コードが異なることが挙げられます。Git で管理されているファイルの改行コードが一致しない

と、差分が発生してしまい、想定した挙動と異なる動作をすることがあります。それを防ぐために、.gitattributes に改行コードを明示しておくことで、安全に開発を進めることができます。

リスト 1.2 のように、ファイルの拡張子に対して、改行コードを指定することができます。

リスト 1.2 .gitattributes

```
1 * text=auto
2 *.sh text eol=lf
```

これも.gitignore と同様に.gitattributes 共有することで、開発メンバー全員が同じ設定を利用することができます。

3.2.3 Makefile

Makefile は、コマンドをまとめて実行するための設定ファイルです。

利点として、開発メンバー全員が同じコマンドを実行することができることが挙げられます。環境構築やテストの実行など、手順が複雑な作業を一人一人が実行した場合、手順の違いによるエラーが発生する可能性があります。それらを防ぐために、Makefile にまとめておくことで、開発メンバー全員が同じコマンドを実行することができ、人的ミスを防ぐことができます。

Makefile の例としては、リスト 1.3 のようなものです。

リスト 1.3 Makefile

```
1 up: ## API とデータベースを起動
2   docker compose -f docker-compose-db.yml -p db up -d
3   docker compose -f docker-compose-api.yml -p api up -d
4
5 build: ## サービスの構築
6   docker compose -f docker-compose-db.yml -p db build
7   docker compose -f docker-compose-api.yml -p api build
8
9 stop: ## サービスを停止
10   docker compose -f docker-compose-db.yml -p db stop
11   docker compose -f docker-compose-api.yml -p api stop
12
13 kill: ## サービスを強制停止
14   docker compose -f docker-compose-db.yml -p db kill
15   docker compose -f docker-compose-api.yml -p api kill
16
17 down: ## サービスの停止とコンテナの削除
18   docker compose -f docker-compose-db.yml -p db down
19   docker compose -f docker-compose-api.yml -p api down
20
21 restart: ## サービスの再起動
22   docker compose -f docker-compose-db.yml -p db restart
23   docker compose -f docker-compose-api.yml -p api restart
```

これらを実行する場合、シェルに

```
$ make up
```

と入力することで、2,3 行目のコマンドが実行されます。リスト 1.3 の例は簡単なものですが、他にも Makefile 内に変数を定義することが出来るため、変更が必要な箇所を変数に置き換えることで、コマンドの変更を容易に行うことができます。

3.3 インフラ回り

3.3.1 GitHub Actions

Github Actions は、GitHub 上で動作する CI/CD ツールです。簡単に言えば、仮想マシン上でコマンドを実行することができるものです。

用途としては、コードの静的解析やテストの実行、デプロイなどが挙げられます。その他にも、Pull Request に対して、テスト内容をコメントすることができる等の GitHub 上での機能を利用することができます。

利用方法としては、リスト 1.4 のように、.github/workflows ディレクトリを作成し、その中に設定ファイルを作成します。

リスト 1.4 .github/workflows/pr.yml

```
1 on:
2   pull_request:
3     types: [opened]
4 name: Pull Request
5 jobs:
6   assignAuthor:
7     name: Assign author to PR
8     runs-on: ubuntu-latest
9     steps:
10      - name: Assign author to PR
11        uses: technote-space/assign-author@v1
```

上記の例では、Pull Request が作成された際に、Pull Request の作成者を Assignee に設定する設定ファイルの例です。

その他にも、リスト 1.5 のように、ssh 鍵を設定することで、リモートサーバーに

アクセスすることができます。

リスト 1.5 .github/workflows/deploy.yml

```
1 name:CI
2 on:
3   push:
4     branches:
5       - main
6 jobs:
7   deploy:
8     runs-on: ubuntu-latest
9     steps:
10      - uses: actions/checkout@v2
11      - name: Install SSH Key for Deploy
12        uses: appleboy/ssh-action@master
13        with:
14          key: ${{ secrets.SK }}
15          host: ${{secrets.SSH_HOST}}
16          username: ${{secrets.SSH_USERNAME}}
17          port: ${{secrets.SSH_PORT}}
18          script: |
21            git pull
```

この他にも、自動で整形して commit してくれるツールなどの様々なツールが存在します。調べてみると面白いかもしれません。

3.4 おわりに

本チャプターでは「リポジトリ作成後に設定しておきたいこと」をご紹介しました。ご紹介したのは一部分ですが、これらを設定しておくことで、開発の効率化や、開発メンバーのミスを防ぐことができます。ぜひ、設定してみてください。

4

これは chapter

4.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

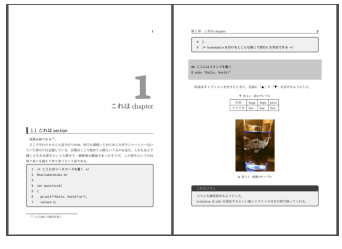
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 4.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 4.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

5 これは chapter

5.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

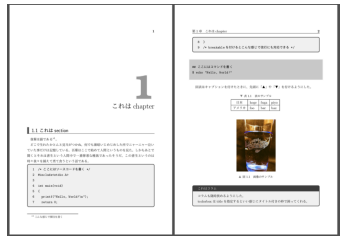
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 5.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 5.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

6 これは chapter

6.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

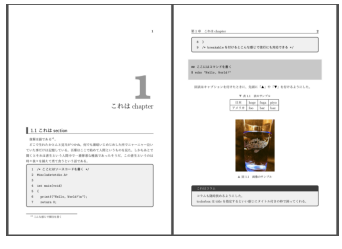
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 6.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 6.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

7
これは chapter

7.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

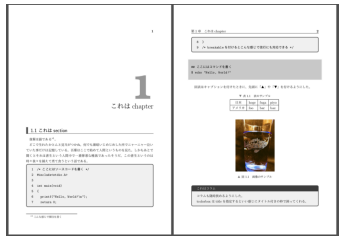
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 7.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 7.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。

8 これは chapter

8.1 これは section

我輩は猫である[*1]。

どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。

```
1  /* ここにはソースコードを書く */
2  #include<stdio.h>
3
4  int main(void)
5  {
```

[*1] こんな感じで脚注を書く

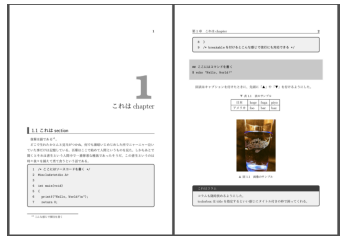
```
    printf("Hello, World!\n");
    return 0;
}
/* breakable を付けるとこんな感じで改行にも対応できる */
```

```
## ここにはコマンドを書く
$ echo "Hello, World!"
```

図表はキャプションを付けたときに、先頭に「▲」や「▼」を付けるようにした。

▼ 表 8.1　表のサンプル

日本	hoge	fuga	piyo
アメリカ	foo	bar	baz



▲ 図 8.1　画像のサンプル

これはコラム

コラムも随時挟めるようにした。
tcolorbox は title を指定するといい感じにタイトル付きの枠で囲ってくれる。