How do slash commands work?

```
presentation.PresentedBy(“Vidhan”)
```

What are slash commands?

Slash commands are a fancy new feature of Discord. If you haven't checked them out yet already, try going to the `#bot-spam` channel in WCS Club, and sending a code snippet, and then running `/run`. Another example was the WCS Checkpoint bot which you likely used to enter the server (Made by yours truly 😉).

Slash commands replace older, <prefix><command> (e.g. !run) type commands.

Why slash commands?

Slash commands replace older, <prefix><command> (e.g. !run) type commands.

Why would Discord do this, you ask?

- Privacy. Older style commands let the bot read **every message** in your server! If the bot was hijacked or coded to be secretly malicious, your messages could be leaked by the developer
- Convenience for both developer and user. This will be covered later in the presentation

```go
dg, err := discordgo.New("Bot " + TOKEN)
```

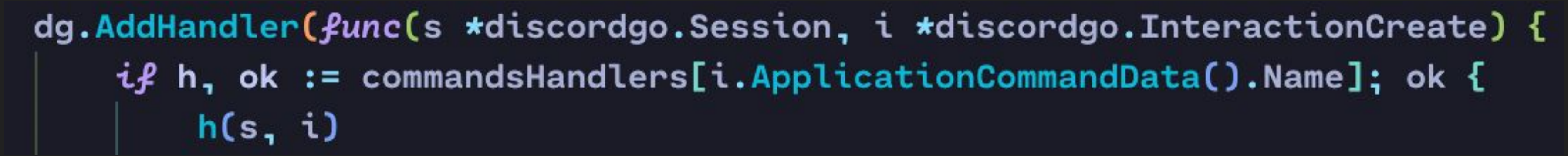
```
dg.AddHandler(...)
```

How does `dg.AddHandler(...)` work?

The `commandsHandlers` map (known as a dictionary in Python) is a map which maps strings (the name of the command) to the code that runs when you type the command in discord!

This may seem odd, as you cannot usually store functions in dictionaries in Python. However, this method we are using is called "functional programming". This is when you treat functions as any other object, and you are able to pass them to other functions. Instead of passing data to a function, you pass a function to data, making it less computing-expensive in many cases.

How does `dg.AddHandler(...)` work?



```go
dg.AddHandler(func(s *discordgo.Session, i *discordgo.InteractionCreate) {
	if h, ok := commandsHandlers[i.ApplicationCommandData().Name]; ok {
		h(s, i)
```

h here is the command. It gets the function from the `commandHandlers` map, and runs it.

For example, say you use `/run`. The `i.ApplicationCommandData().Name` would be `"run"`. It would then check the `commandsHandlers` map, and if ok evaluates to true (ok is Go's way of returning whether or not the key wanted is in the map's keys, and only if it is true, we run the function. If it is not true, the code will not run, or else it would error as h would not exist at all.)

```
dg.ApplicationCommandBulkOverwrite(...)
```

How does `dg.ApplicationCommandBulkOverwrite(...)` work?

This actually tells Discord the valid commands which are available to the bot.

Think about a time before slash commands… We had to use some !help command which would half of the time open up a webpage without any search functionality, and then even if you found the command, you would be confused what the arguments were! The bot would also randomly pick up sentences which happened to start with the prefix it used, making it annoying if it happened in the middle of the conversation.

On the developer side, it was arguably just as bad. Developers had to manually validate argument types, as well as implement and update `help` commands for the bot.

This is what a strictly-defined slash command arrays fix. Now, we can tell Discord the exact format of our commands, so we and the users do not have to worry about making sure of the correct format types!

`dg.ApplicationCommandBulkOverwrite(...)` takes in an array of commands, and then sends it to Discord so the users can see them.

Conclusion

Now that we understand each part, let's put it all together.

First, we initialize the bot. We then add handlers for each command, which are essentially functions which are run when a command is run. We then add the valid commands, as well as their valid parameters, so there is no confusion.

Now, how would this work on the user side?

They would type `/` in their discord bar, bringing up all the valid commands and their parameters, which Discord received from the `commands` array. They would run a command, and then our bot would check if the command existed in our `commandsHandlers` map. If so, it would run the function in the map, and that function would reply to the message sent by the user, finishing the process.