DISCORD BOT WORKSHOP AND BOTATHON 2024

Written by Holly
High quality assets by Leo and Lee

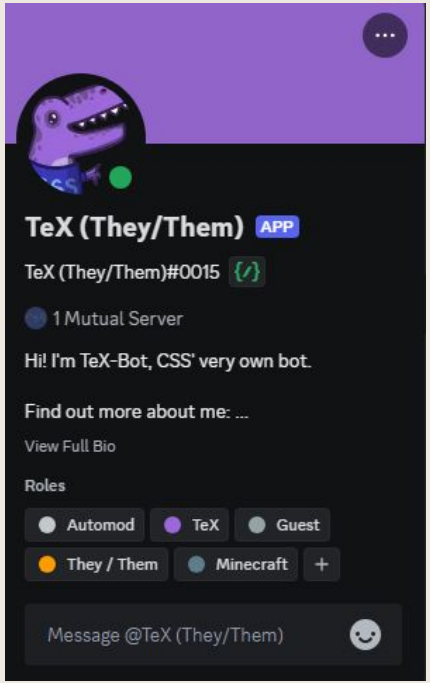
Workshop Contents

- Why make a Discord bot?
- Creating the bot on Discord
- Installing discord.py
- Listening to events
- Legacy commands
- Slash commands
- Cogs
- More advanced components
- Common Issues
- Ideas ahead of the workshop

Why make a Discord bot?

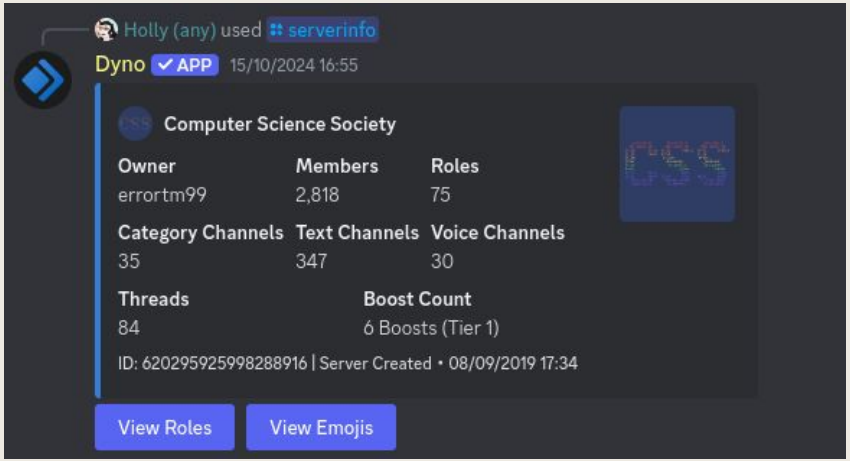
Discord bots are an easy way to add some flare to your server.

- Can be used for moderation, raid protection or logging
- Can also have an interactive game that users win points for
- If you have a support server, you could make a FAQ or ticket bot
- The possibilities are endless!



Popular Bots

Nice image!



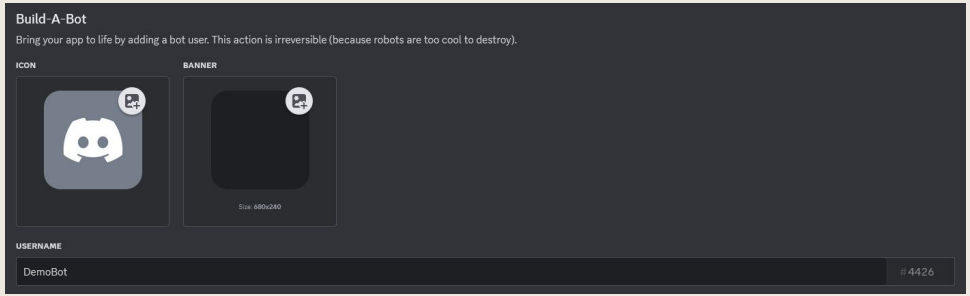
Discord API Wrappers

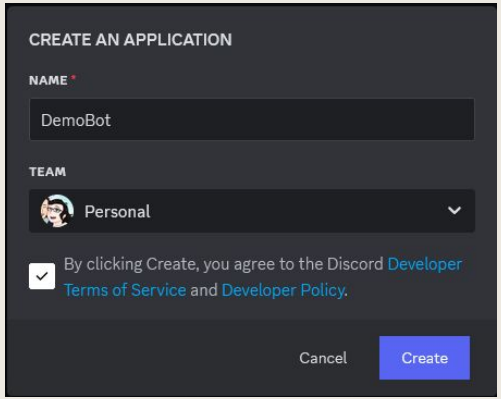
The Discord API is a RESTful API with WebSockets for events, but there's a lot of wrapper APIs making it easy to write bots in different languages. We will using discord.py for this workshop, but you can use any wrapper tomorrow.

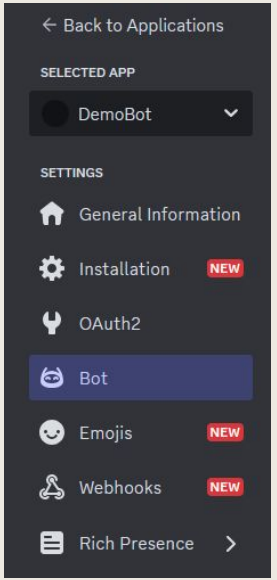
- discljord (Closure)
- discordcr (Crystal)
- nyxx (Dart)
- discord.net (.NET)
- Coxir (Elixir)
- discordgo (Go)
- JDA (Java)
- discord.js (JS)
- Kord (Kotlin)
- Discordia (Lua)
- Dimscord (Nim)
- Discord.php (PHP)
- **discord.py (Python)**
- discordrb (Ruby)
- Serenity (Rust) (L)
- Ackcord (Scala)
- Swiftcord (Swift)

Creating the Bot on Discord

- Navigate to https://discord.com/developers/applications
- Click on the “New Application” button in the top right
- Give it a name!
- Create it, then navigate to “Bot”.
- You can set the name, icon, etc.
  - Yes bots still have discriminators lmao

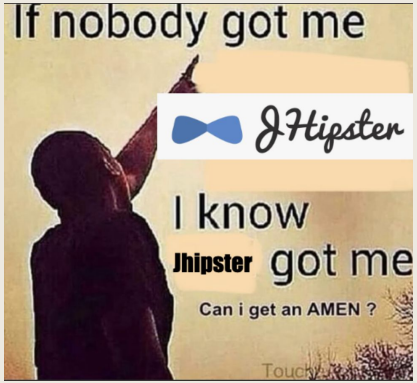


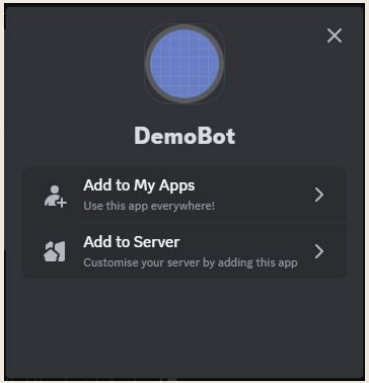


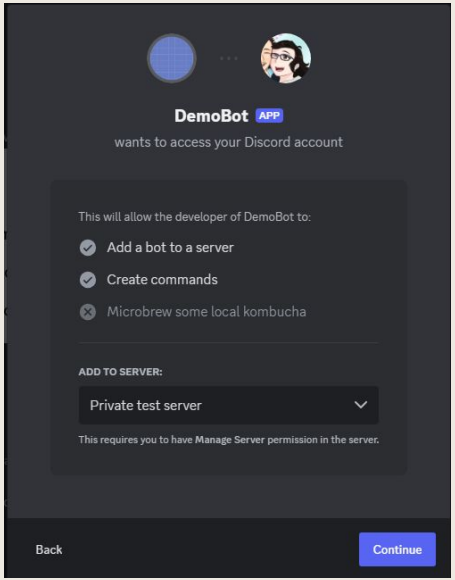


Adding the Bot to a Server

- Go to “Installation”
- Select permissions that your bot will have access to
- Copy the installation link and paste it in a browser
- Choose the server to add the bot to





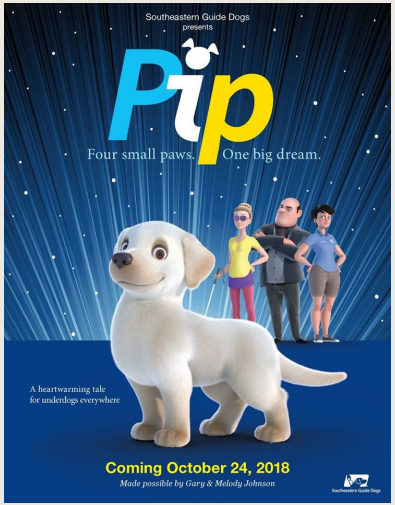


Tools for Bot Development

- Python 3.8 to 3.12
  - This workshop assumes some knowledge in Python
  - Requires pip with it
- PyCharm Ultimate or VSCode

Installing discord.py

- Requires Python ⩾ 3.8
- Create a virtualenv for your bot and activate it
  - `python3 -m venv venv`
  - Linux: `source venv/bin/activate`
  - Windows: Install Linux
  - `venv\Scripts\activate.bat`
- Install discord.py via pip
  - `pip install -U discord.py`
  - With voice support: `pip install -U discord.py[voice]`



Running the Bot

- You'll need to supply the bot's token
- Set the required intents
- Can declare events
- Then run the bot
  - Python3 bot.py
- Don't commit tokens to repos - use a package like dotenv to carry across tokens

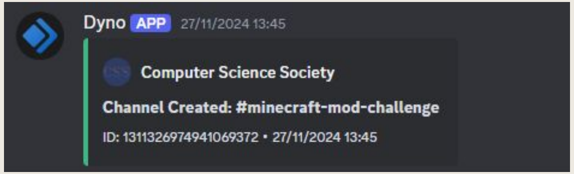
```python
import os

import discord
import dotenv

dotenv.load_dotenv()

intents = discord.Intents.default()
intents.message_content = True

client = discord.Client(intents=intents)


@client.event
async def on_ready():
    print("Hello, world!")

client.run(os.environ["BOT_TOKEN"])

```

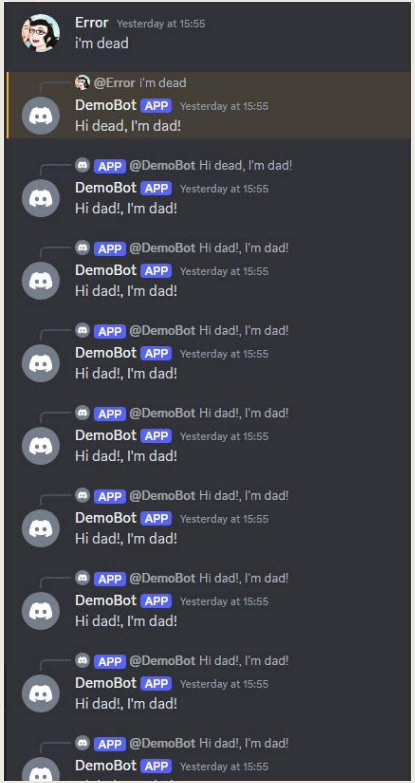
Running the Bot - Securing your Token

- If someone finds your token, they can use it to run code through your bot
- When testing/running locally:
  - Use a .env file to store the token and ensure it's ignored by Git
- When running on a hosting service:
  - Use advantage of the service's "vault" for storing secrets (if they offer one), and review documentation for accessing those secrets

Listening to Events

- A lot of things happen on Discord - users joining servers, reactions being added, buttons being pressed, etc.
- You may choose to listen to some events for the bot to do its job
- Example uses:
    - Automoderation
    - Auto responses
    - Event logging
    - Starboards





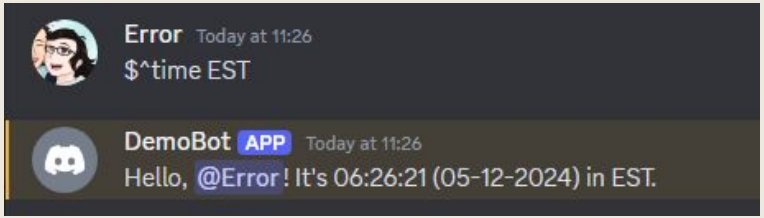
Listening to Events - Implementation

- Uses the @client.event decorator
  - “Client” may vary depending on the variable name used for the client instance
- Function names are specific - e.g. on_message, on_guild_join, etc.
- Must be a coroutine (i.e. async def function_name())
- Event reference: https://discordpy.readthedocs.io/en/stable/api.html#event-reference

```python
import re

im_regex = re.compile("(?i)i('m| am) (.+)")

@client.event
async def on_message(message: Message):

    # Don't reply to ourselves
    if message.author == client.user:
        return

    # See if a user said "i'm ..." and reply "hi ..., i'm dad!"
    matches = im_regex.findall(message.content)
    for match in matches:
        await message.reply(f"Hi {match[1]}, I'm dad!")
```

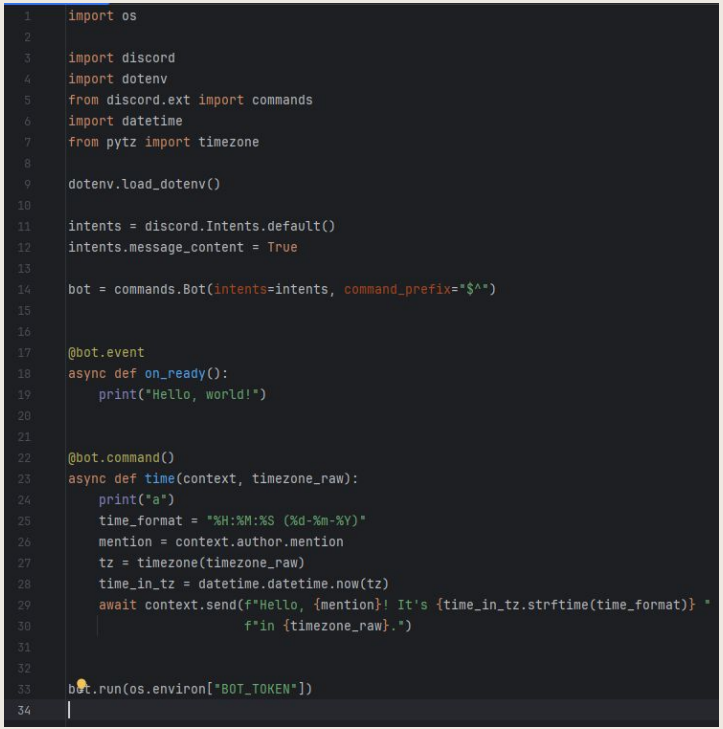
Legacy Commands

- Inbuilt feature of discord.py - old-school commands
- Requires message_content intent
- Considered an extension of the main API
  - Your code will look a bit different when initialising the bot to use these



Legacy Commands - Implementation

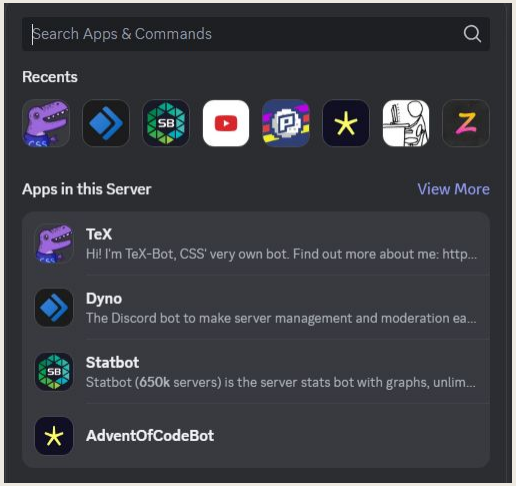
- Requires a different client implementation
  - Instead of discord.Client, we use commands.Bot - of which has the ability to register commands.
- Use @bot.command decorator to declare a command + the name
- Run the bot
  - I used the pytz package to fetch the timezone requested - you can use any pip package to support functions of your bot.

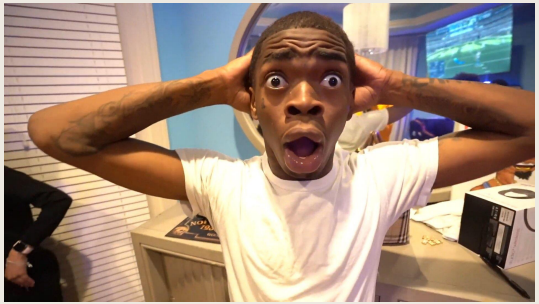


```python
import os

import discord
import dotenv
from discord.ext import commands
import datetime
from pytz import timezone

dotenv.load_dotenv()

intents = discord.Intents.default()
intents.message_content = True

bot = commands.Bot(intents=intents, command_prefix="$^*")


@bot.event
async def on_ready():
    print("Hello, world!")


@bot.command()
async def time(context, timezone_raw):
    print("a")
    time_format = "%H:%M:%S (%d-%m-%Y)"
    mention = context.author.mention
    tz = timezone(timezone_raw)
    time_in_tz = datetime.datetime.now(tz)
    await context.send(f"Hello, {mention}! It's {time_in_tz.strftime(time_format)} "
                       f"in {timezone_raw}.")


bot.run(os.environ["BOT_TOKEN"])
```

Slash Commands

- Commands hooking directly into Discord's commands feature
- Allows argument validation/enforcement
- The modern approach to commands - allows to send private responses and doesn't require the message_content intent





Slash Commands - Implementation (Client)

- If you're testing commands, specify your test server's ID
  - Otherwise, you'll wait at least an hour for the commands to show up
- Create a class for the client to override the setup_hook function
- Create a tree object using app_commands.CommandTree
- In the setup_hook function, copy and sync the commands to the test server

```python
dotenv.load_dotenv()

TEST_SERVER = discord.Object(id=254987557459329025)


1 usage
class MyClient(discord.Client):

    def __init__(self):
        intents_list = discord.Intents.default()
        intents_list.message_content = True
        super().__init__(intents=intents_list)

        self.tree = app_commands.CommandTree(self)

    async def setup_hook(self) -> None:
        self.tree.copy_global_to(guild=TEST_SERVER)
        await self.tree.sync(guild=TEST_SERVER)


client = MyClient()
```

Slash Commands - Implementation (Command)

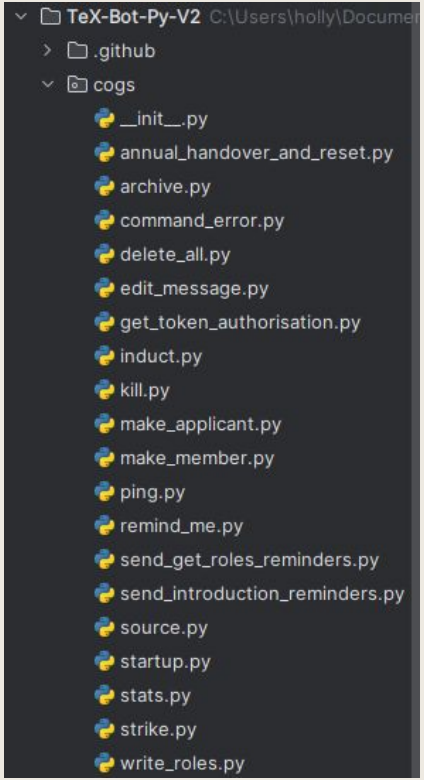
- Use client.tree.command decorator to declare a slash command
- First argument is the interaction, and second argument reflects the required command argument
- When you want to respond, use interaction.response.send_message()
- Set ephemeral=True to make it visible to just the command sender

```python
@client.tree.command(description="Play Rock Paper Scissors against the bot.")
@app_commands.describe(
    item="The item to select."
)
async def rps(interaction: discord.Interaction, item: str):
    valid_items = ["rock", "paper", "scissors"]
    chosen_user_item = item.lower()
    if item not in valid_items:
        await interaction.response.send_message(f"You can't use **{item}**!", ephemeral=True)

    chosen_bot_item = random.choice(valid_items)

    user_item_index = valid_items.index(chosen_user_item)
    bot_item_index = valid_items.index(chosen_bot_item)

    if bot_item_index == user_item_index:
        await interaction.response.send_message(
            f"It's a draw! {interaction.user.mention} picked **{chosen_user_item}**, I picked **{chosen_bot_item}**!")

    # If the user chose the item in front (e.g. bot -> scissors, user -> rock), the user wins
    if valid_items[(bot_item_index + 1) % 3] == item:
        await interaction.response.send_message(
            f"You won! {interaction.user.mention} picked **{chosen_user_item}**, I picked **{chosen_bot_item}**!")
    else:
        await interaction.response.send_message(
            f"You lost! {interaction.user.mention} picked **{chosen_user_item}**, I picked **{chosen_bot_item}**!")

client.run(os.environ["BOT_TOKEN"])
```

PyCharm says send_message is not a function… it is!

Cogs

- A discord.py/Pycord specific feature - mainly for larger bots
- Allows you to add commands/event listeners in different files
- A modular approach to bot development



Cogs - Implementation

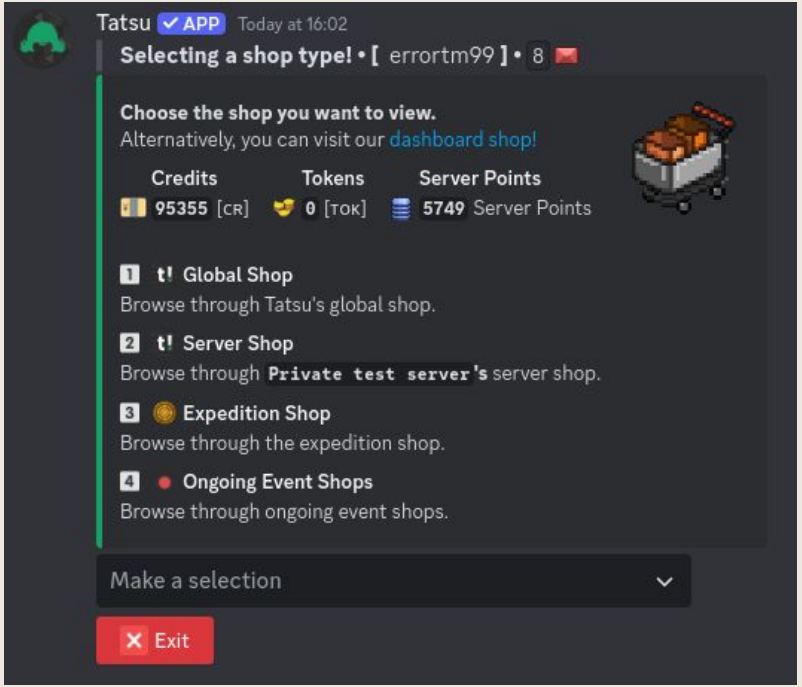
- Create a class extending commands.Cog
- Add your listeners and commands to the cog
- In your main file, add the cog
- To disable it, remove it again using its name
  - The view keyword we'll see again later!

```python
class MyClient(discord.ext.commands.Bot):

    def __init__(self):
        intents_list = discord.Intents.default()
        intents_list.message_content = True
        super().__init__(intents=intents_list, command_prefix="$^*")


    async def setup_hook(self) -> None:
        await self.add_cog(PronounSelectCog())

        self.tree.copy_global_to(guild=TEST_SERVER)
        await self.tree.sync(guild=TEST_SERVER)
```

```python
class PronounSelectCog(commands.Cog):

    @app_commands.command()
    async def send_pronouns(self, interaction: discord.Interaction):
        await interaction.response.send_message("Pick your pronouns: ", view=PronounSelectView())
```

Embeds

- A messaging feature only available to bots
- Allows data to be structured in a neat, condensed manner
- In discord.py, it’s a mutable data structure



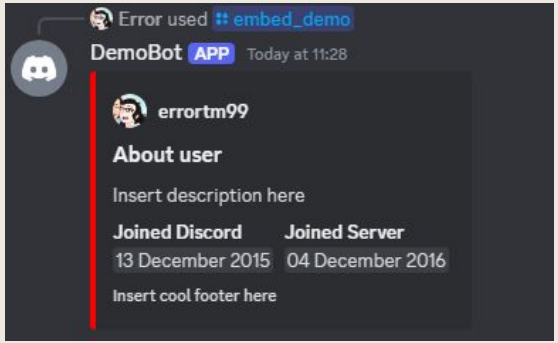
Embeds - Structure

- Author name + image
- Title (can be a masked link)
- Thumbnail (top right)
- Description
- Fields (inline or not)
- Image (at the bottom)
- Footer

Image credit: https://anidiots.guide/first-bot/using-embeds-in-messages/

Embeds - Implementation

- Create an embed object, specifying the title, description and colour
- Add fields
- Set the image/footer after creation
- Send it using the embed argument




```python
@client.tree.command()
async def embed_demo(interaction: discord.Interaction):
    embed = discord.Embed(title="About user", description=f"Insert description here", color=0xff0000)
    embed.set_author(name=interaction.user.name, icon_url=interaction.user.avatar.url)
    embed.set_footer(text="Insert cool footer here")
    embed.add_field(name="Joined Discord", value=f"<t:{int(interaction.user.created_at.timestamp())}:D>", inline=True)
    embed.add_field(name="Joined Server", value=f"<t:{int(interaction.user.joined_at.timestamp())}:D>", inline=True)

    await interaction.response.send_message(embed=embed)

```

Interactions

Interactions are events where users interact with the bot, either through:

- Slash commands
- Forms (Modals)*
- Buttons
- Dropdown menus (Select menus)

* For modals, they have a limit of 5 items at a time - stupid choice but oh well

Interactions - Sending Buttons

- Create a class extending discord.ui.View
- For each button, add a function with the decorator @discord.ui.button
- Declare the style and label of the button
- Have the method body be the code run when the button is clicked


```
1 usage
class NukeServerView(ui.View):

    @ui.button(label="Yes", style=discord.ButtonStyle.green)
    async def confirm_nuke(self, interaction: discord.Interaction, button: discord.ui.Button):
        await interaction.response.send_message("Sure thing, nuking server...")

    @ui.button(label="No", style=discord.ButtonStyle.red)
    async def cancel(self, interaction: discord.Interaction, button: discord.ui.Button):
        await interaction.response.send_message("Ok! We'll put things off until a later date.")
```

```
@client.tree.command()
async def nuke_server(interaction: discord.Interaction):
    await interaction.response.send_message("Oh boy, are you sure you want to nuke the server?", view=NukeServerView())
```

Interactions - Sending Dropdowns

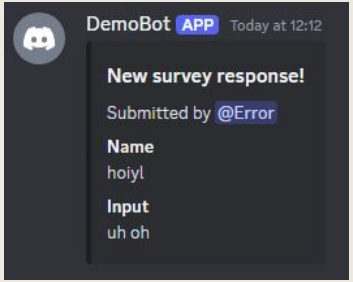
- Create a class extending discord.ui.Select
- Create a class extending discord.ui.View
- In the __init__ function, add the select menu
- In the select class, override the callback function and handle the response
- Send the view in a message

```
2 usages
class PronounSelectCog(commands.Cog):

    @app_commands.command()
    async def send_pronouns(self, interaction: discord.Interaction):
        await interaction.response.send_message("Pick your pronouns: ", view=PronounSelectView())
```

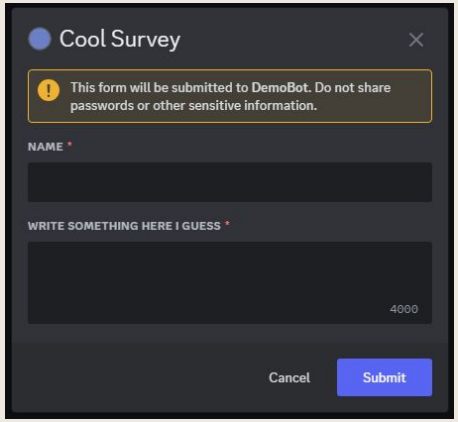
```
class PronounSelect(discord.ui.Select):

    def __init__(self):

        options = [
            discord.SelectOption(label="He/Him"),
            discord.SelectOption(label="She/Her"),
            discord.SelectOption(label="They/Them")
        ]
        super().__init__(placeholder="Select your pronouns", min_values=0, max_values=3, options=options)

    async def callback(self, interaction: Interaction[ClientT]) -> Any:
        guild = interaction.guild
        removing_roles = {
            "He/Him": guild.get_role(HE_ROLE.id),
            "She/Her": guild.get_role(SHE_ROLE.id),
            "They/Them": guild.get_role(THEY_ROLE.id)
        }
        for value in self.values:
            if value in removing_roles:
                role = removing_roles.pop(value)
                await interaction.user.add_roles(role)

        for role in removing_roles.values():
            await interaction.user.remove_roles(role)

        await interaction.response.send_message("Updated your pronouns!", ephemeral=True)


1 usage
class PronounSelectView(discord.ui.View):
    def __init__(self):
        super().__init__()
        self.add_item(PronounSelect())
```

Interactions - Modals

- Create a class extending the Modal class
- Specify the title and inputs
- Override the on_submit function to note answers




```python
@client.tree.command()
async def take_survey(interaction: discord.Interaction):
    await interaction.response.send_modal(Survey())
```




```python
1 usage
class Survey(ui.Modal, title="Cool Survey"):
    name = ui.TextInput(label="Name")
    input = ui.TextInput(label="Write something here I guess", style=discord.TextStyle.paragraph)

    async def on_submit(self, interaction: Interaction) -> None:
        server = client.get_guild(TEST_SERVER.id)
        channel = server.get_channel(SURVEY_CHANNEL.id)

        embed = discord.Embed(title="New survey response!", description=f"Submitted by {interaction.user.mention}")
        embed.add_field(name="Name", value=self.name.value, inline=False)
        embed.add_field(name="Input", value=self.input.value, inline=False)

        await channel.send(embed=embed)
        await interaction.response.send_message("Thank you for filling out the survey!", ephemeral=True)
```

Common Issues

- Slash commands take light years to register
  - They will instantly register if you specify a server/Guild
- Slash command/interaction did not respond in time
  - The normal timeout for these is 5 seconds
  - After this point, Discord will tell the user the app did not respond
  - If you cannot finish something in 5 seconds, you can defer the interaction to acknowledge it
- Bot not starting due to accessing privileged intent
  - Make sure it’s enabled in the developer portal!

Ideas ahead of time

- The Discord bot scene was already rich 3 years ago…
- … but now, you have even more options to change the experience.
- What will you choose to do tomorrow?
  - Recreate a popular game, but only through a Discord bot?
  - An image-manipulating bot?
  - Something making use of Matplotlib?
  - A bot that can change the colour of the RGB strips in your room?
- Best of luck for tomorrow!

Further Learning

- Check discord.py's examples folder: https://github.com/Rapptz/discord.py/tree/v2.4.0/examples
- Explore the API references for discord.py:
  - https://discordpy.readthedocs.io/en/stable/api.html
  - https://discordpy.readthedocs.io/en/stable/interactions/api.html
  - https://discordpy.readthedocs.io/en/stable/ext/commands/api.html
- Timed tasks - bonus feature: https://discordpy.readthedocs.io/en/stable/ext/tasks/index.html


Z z