



# Minicurso de Bots Discord com Python

Aula 2



## Introdução aos comandos

Na última aula utilizamos *startswith* para criar um comando, dentro do evento *on\_message*.

```
@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if message.content.lower().startswith('!oi'):
        await message.channel.send(f'Oii {message.author.mention}!')
```



## Introdução aos comandos

Para passar argumentos aos comandos, teríamos que fazer um parser para cada comando, deixando o código complicado.

Por isso iremos utilizar uma estrutura diferente.



## Estrutura básica de um comando

Invés de utilizarmos `discord.Client` para definição do cliente, usaremos `commands.Bot`

```
discord.Client(intents=discord.Intents.all())
```

```
from discord.ext import commands  
commands.Bot(intents=discord.Intents.all(), command_prefix='seu prefixo')
```



## Criação de um comando sem argumentos

Agora para criação de um comando, utilizamos um decorador de funções.

```
if message.content.lower().startswith('!oi'):  
    await message.channel.send(f'Oii {message.author.mention}!')
```

Decorador

`@client.command()`

`async def oi(context):`

Nossa função, que por ser assíncrona, é chamada de corrotina

`await context.channel.send(f'Oii {context.author.mention}!')`



## Criação de um comando com argumentos

Para comandos onde é necessário fornecer argumentos, apenas precisamos modificar a declaração da função

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```



## Criação de um comando com argumentos

*nome* é uma variável que representa o argumento recebido quando a função foi chamada, por não termos declarado tipo, ela por padrão é uma *str*.

O comando pode ser utilizado como `,oi Natã`

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```



## Criação de um comando com argumentos

Note que 'Natã' só se tornou argumento da função pois havia um espaço entre o invocador 'oi' e o argumento.

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```





## Criação de um comando com argumentos

O comando ,oi Natã Schmitt funcionará como esperado?

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```




## Criação de um comando com argumentos

Não, pois inserimos 2 argumentos para *oi*, que esperava receber apenas um.

Há duas maneiras de corrigir este problema:

Alterando o comando para `,oi "Natã Schmitt"`,  
ou modificando a definição da função.

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```



Criação de um comando com múltiplos argumentos

Como podemos alterar nosso comando para que ,oi Natã Schmitt funcione, mas com um número qualquer de argumentos?

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```



## Solução

nome agora é uma lista de *str*, apenas juntamos no final

```
@client.command()
async def oi(context, *nome):
    await context.channel.send(f'Oii {" ".join(nome)}!')
```



## Criação de um comando com argumentos

Se quiséssemos receber um inteiro ao invés de string, podemos fazer:

```
@client.command()
async def par(context, num):
    if not num.isdigit():
        await context.reply('Digite um número!')
    else:
        num = int(num)
        await context.reply(f'{num} é {"PAR" if num%2 == 0 else "ÍMPAR"}!')
```



Criação de um comando sem argumentos

Agora, no discord do minicurso, digite o comando *,prefixo* no chat #general.

O bot irá designar um prefixo para você utilizar em seu bot.

Copie, modifique e execute o código no #aula-2, colocando seu prefixo e seu token.

Exemplo:

```
commands.Bot(intents=discord.Intents.all(), command_prefix=';')
```



## Tratamento de erros

Com o comando de antes, o que acontece se enviarmos apenas ,oi?

```
@client.command()
async def oi(context, nome):
    await context.channel.send(f'Oii {nome}!')
```



# Tratamento de erros

O usuário não terá nenhuma resposta, e no terminal, horrendamente aparecerá isto:

```
2022-12-11 02:18:35 ERROR discord.ext.commands.bot Ignoring exception in command oi
Traceback (most recent call last):
  File "/home/natas/bot_minicurso/venv/lib/python3.8/site-packages/discord/ext/commands/bot.py", line 1349, in invoke
    await ctx.command.invoke(ctx)
  File "/home/natas/bot_minicurso/venv/lib/python3.8/site-packages/discord/ext/commands/core.py", line 1015, in invoke
    await self.prepare(ctx)
  File "/home/natas/bot_minicurso/venv/lib/python3.8/site-packages/discord/ext/commands/core.py", line 932, in prepare
    await self._parse_arguments(ctx)
  File "/home/natas/bot_minicurso/venv/lib/python3.8/site-packages/discord/ext/commands/core.py", line 839, in _parse_arguments
    transformed = await self.transform(ctx, param, attachments)
  File "/home/natas/bot_minicurso/venv/lib/python3.8/site-packages/discord/ext/commands/core.py", line 691, in transform
    raise MissingRequiredArgument(param)
discord.ext.commands.errors.MissingRequiredArgument: nome is a required argument that is missing.
```





# Tratamento de erros

Como evitar?

Há diversas maneiras...



## Tratamento de erros

A mais simples para nosso caso, é utilizarmos o conhecimento anterior. Se a lista em *nome* estiver vazia, o usuário não digitou seu nome no comando

```
@client.command()
async def oi(context, *nome):
    if len(nome) == 0:
        await context.reply('Ei, cadê seu nome? Eu sei que você tem!! 🐼')
    else:
        await context.reply(f'Oii {" ".join(nome)}!')
```



## Desafio

Após executar o bot com sucesso e verificar que está respondendo corretamente com o prefixo, implemente um comando que some dois ou mais números inteiros.

Ex.: ,soma 3 5 e o bot responderá 8

,soma 3 5 8 -2 e o bot responderá 14