Actions & Python

Como utilizar Actions e Python no RASA

Quem somos nós?

Rafa

Debs

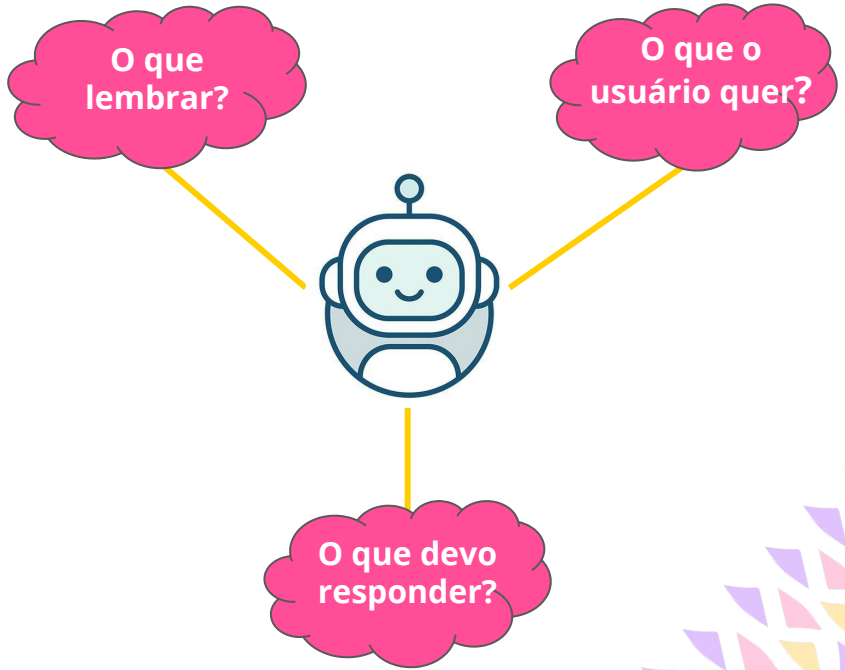
ACTIONS

Como criá-las? Do que se alimentam?

Relembrando...

Como o bot funciona

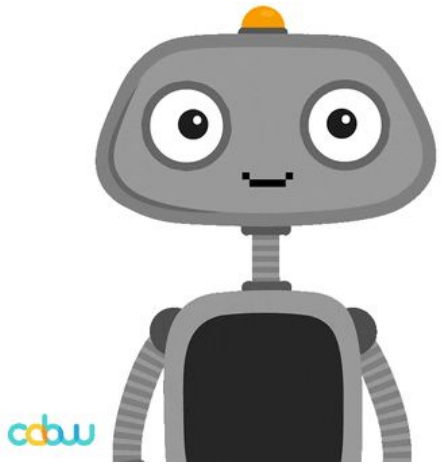
- Intents: o assunto que será tratado;
- **Actions:** as respostas que serão dadas;
- **Slots**: memória do bot.



O que é e para que pode ser usada

- Actions são coisas que seu bot executa em resposta à mensagem recebida.
- Enviar uma mensagem de texto, uma imagem, chamar uma API, inserir um evento em um calendário....

Tudo e mais um pouco!!!



Tipos de actions

- Responses ou Utterance actions
- Custom actions
- Forms
- Default actions

As comuns (intent + utter)

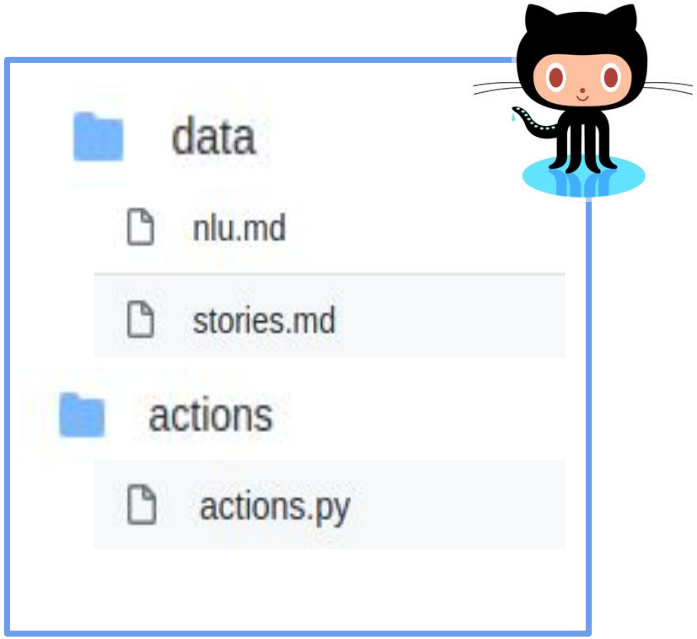
Personalizadas, pode-se criar qualquer coisa

É um tipo especial de custom actions, que lida com lógica de negócios

Actions que são padrão, já existem, é só usar

Como criar uma action

- Arquivo **domain.yml**
- Arquivo **actions.py**
  - Arquivo **stories.md** se for uma utterance action



Como criar uma action

- Arquivo **domain.yml**
- Arquivo **actions.py**
  - Arquivo **stories.md** se for uma utterance action

Listar as actions no domain

domain.yml

```yaml
intents:
    - harry_potter
    - friends

entities:
    - filmes
    - series

responses:
    utter_harry_potter:
    - text: Adoro HP!!
    utter_friends:
    - text: I'll be there for you...

actions:
    - action_ola_mundo
```

Como criar uma action

- Arquivo **domain.yml**
- Arquivo **actions.py**
  - Arquivo **stories.md** se for uma utterance action

Listar as actions nas stories

stories.md

```
## harry_potter
* harry_potter
    - utter_harry_potter

## friends
* friends
    - utter_friends

## ola mundo path
* ola_mundo
    - action_ola_mundo
```

Métodos da classe Action

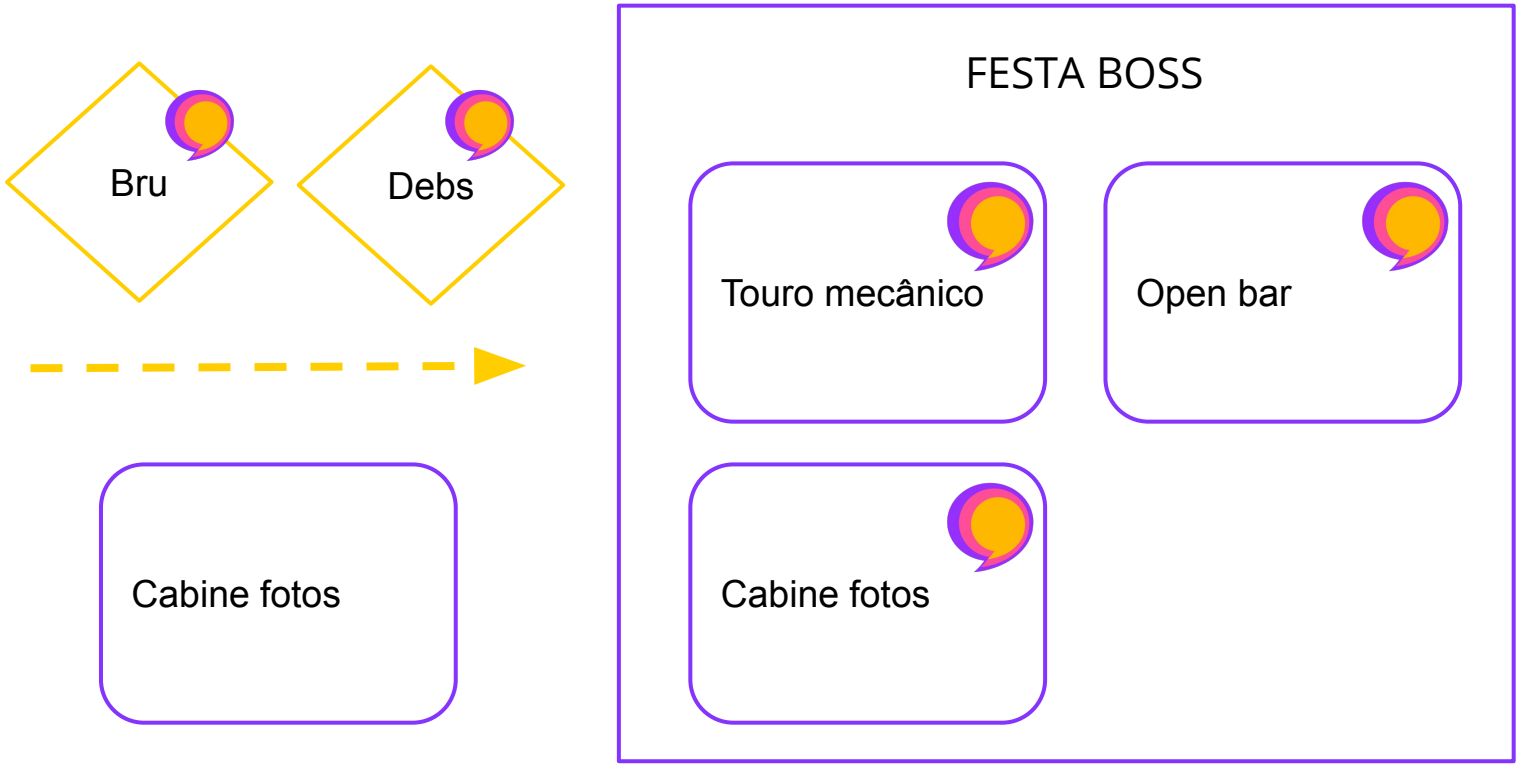
```
def name (self)
def run (self, dispatcher, tracker, domain)
```

O método name( ) recebe como parâmetro de entrada:

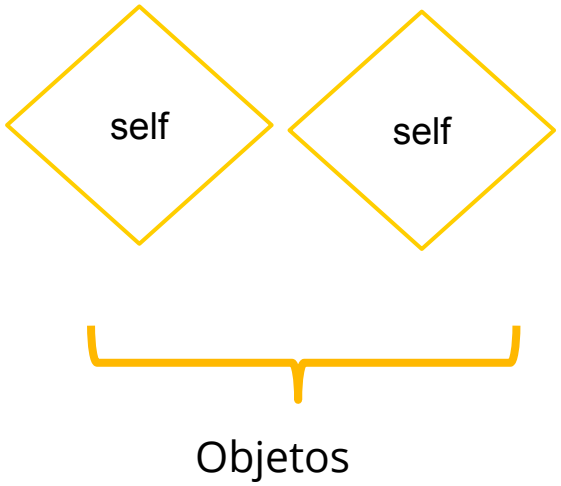
- Self : ele existe em todos os métodos das classes em python e representa o próprio objeto.

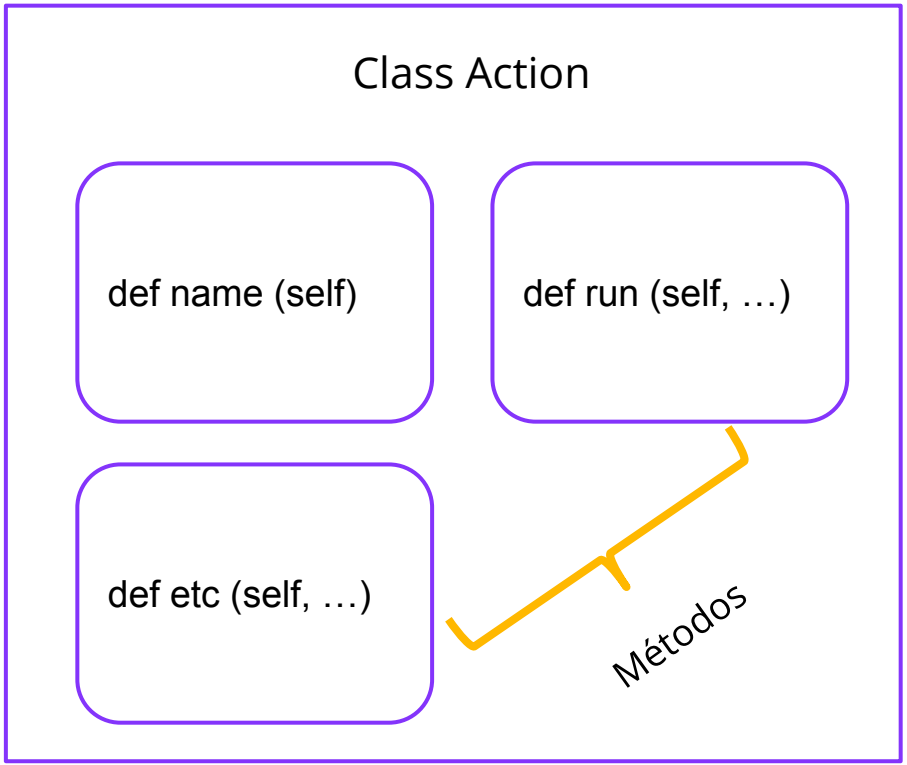
Retorna o nome da sua action.

Analogia de Orientação a Objetos



Analogia de Orientação a Objetos





Métodos da classe Action

```
def name (self)
def run (self, dispatcher, tracker, domain)
```

O método run( ) recebe 4 parâmetros como entrada:

- Self : identificando que o objeto pode utilizar;
- Dispatcher : usado para enviar mensagens de volta ao usuário;
- Domain : o arquivo yml contendo a listagem de tudo do bot;
- Tracker : rastreador; acessa a memória do bot nas custom actions.

Retorna uma lista de eventos.

Exemplo de action

```
class ActionOlaMundo(Action):
    def name(self) -> Text:
        return "action_ola_mundo"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        try:
            dispatcher.utter_message("Olá Mundo!! A BOSS é t.u.d.o")
        except ValueError:
            dispatcher.utter_message(ValueError)
        return []
```

Tracker & Slots

- **Tracker**: usado na custom action para acessar a memória do bot. Possui vários métodos e vários atributos;
- **Slot**: memória do bot que funciona como armazenamento de valor-chave
  - Definido no domain;
  - Definido no NLU;
  - Definido por botões;
  - Definido em actions.

Tracker & Slots

```python
class ActionTelefone(Action):
    def name(self) -> Text:
        return "action_telefone"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:

        telefone = tracker.get_slot('telefone')

        try:
            dispatcher.utter_message("O seu telefone é {}?".format(telefone))
        except ValueError:
            dispatcher.utter_message(ValueError)
        return [SlotSet("telefone", telefone)]
```

PYTHON

O que é importante saber para fazer suas *actions*

Python

Linguagem Open Source (link pro repo) criada em 1992 por Guido van Rossum

Características:

- Alto nível
- Estilo de tipagem **dinâmica** e **forte**

Actions.py

```python
# This files contains your custom actions which can be used to run
# custom Python code.
#
# See this guide on how to implement these action:
# https://rasa.com/docs/rasa/core/actions/#custom-actions/

from typing import Any, Text, Dict, List

from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet


class ActionTeste(Action):
    def name(self) -> Text:
        return "action_teste"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        try:
            dispatcher.utter_message("Mensagem enviada por uma custom action.")
        except ValueError:
            dispatcher.utter_message(ValueError)
        return []
```

rasa-ptbr-boilerplate/bot/actions/actions.py

Como crio um programa em python?

1. Criamos um arquivo com qualquer nome e que termine com a extensão ".py"

exemplo.py

```
1 print("Olá, pessoal da BOSS")
2 print("Vamos aprender juntas!")
3
```

2. Escrevemos código em python dentro dele

```
~/MyProjects
▶ python3 exemplo.py

Olá, pessoal da BOSS
Vamos aprender juntas!
```

3. No terminal, executamos o arquivo com o comando "python"

(ou "python3" para especificar a versão de python que quermos usar)

Como crio um programa em python?

4. Também podemos programar usando a shell do python

Python Shell

```
~/MyProjects
▶ python3
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Olá, pessoal da BOSS")
Olá, pessoal da BOSS
>>> print("Vamos aprender juntas!")
Vamos aprender juntas!
>>>
>>> exit()

~/MyProjects
▶
```

Usando variáveis

```
1 minha_variavel = 3
2 minha_variavel = "três"
3
```

Não precisamos **especificar qual o tipo** da nossa variável ao criá-la. O python "adivinha" sozinho.

Podemos **mudar o tipo** da nossa variável no meio do nosso código sem problemas

Por padrão, nomeamos as variáveis com ***snake_case*** (separando as palavras com underlines)

Usando variáveis [tipos]

- int
- float
- str
- bool

Python shell

```
>>> x = 3
>>> y = 0.556
>>> texto = "um texto qualquer"
>>> sou_linda = True
>>>
```

True = Verdadeiro
False = Falso

Algumas estruturas de dados

- list
- dict
- classes

Python shell

```
>>> lista = [1, 2, 3, 4]
>>> lista[0]
1
>>> lista[0] = "um"
>>> print(lista)
['um', 2, 3, 4]
>>>
```

```
>>> dicionario = {"nome": "Debs", "qtd_olhos": 2}
>>> dicionario["nome"]
'Debs'
>>> dicionario["nome"] = "Débora"
>>> print(dicionario)
{'nome': 'Débora', 'qtd_olhos': 2}
>>>
```

Operadores aritméticos

- **+** (soma)
- **-** (subtração)
- ***** (multiplicação)
- **/** (divisão)
- **//** (divisão inteira)
- **%** (resto da divisão)

Funcionam que nem na matemática que aprendemos na escola

Python shell

```
>>> 5 / 2
2.5
>>> 5 // 2
2
>>> 5 % 2
1
>>>
```

Operadores aritméticos

- **+** (soma)
- **-** (subtração)
- ***** (multiplicação)
- **/** (divisão)
- **//** (divisão inteira)
- **%** (resto da divisão)

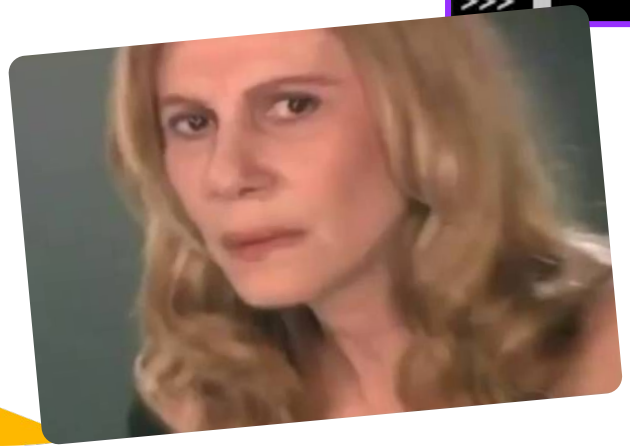
E a gente pode sair fazendo conta com as variáveis que a gente quiser?

Operadores aritméticos

- **+** (soma)
- **-** (subtração)
- ***** (multiplicação)
- **/** (divisão)
- **//** (divisão inteira)
- **%** (resto da divisão)

⚠️ ATENÇÃO ⚠️

```
>>> 3 * "batata"
'batatabatatabatata'
>>> 3 + "vish"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```



Operadores relacionais (comparação)

- `>` (maior que)
- `<` (menor que)
- `>=` (maior ou igual que)
- `<=` (menor ou igual que)
- `!=` (diferente de)
- `==` (igual a)

Python shell

```
>>> 4 > 3
True
>>> "texto" == "Texto"
False
>>>
```

Operadores lógicos (booleanos)

- **and** ("e" - é verdadeiro apenas se tudo verdade)
- **or** ("ou" - é verdadeiro se no mínimo uma das condições é verdade)
- **not** ("não" - retorna o oposto)

Python shell

```
>>> chove = True
>>> troveja = False
>>> chove and troveja
False
>>> chove or troveja
True
>>> not troveja
True
>>>
```

Condicionais

⚠️ Indentação

exemplo.py

```
1 x = 0
2
3 if x > 0:
4     print("x é maior que 0")
5 elif x == 0:
6     print("x é igual a 0")
7 else:
8     print("x é menor que 0")
9
```

```
~/MyProjects
▶ python3 exemplo.py

x é igual a 0
```

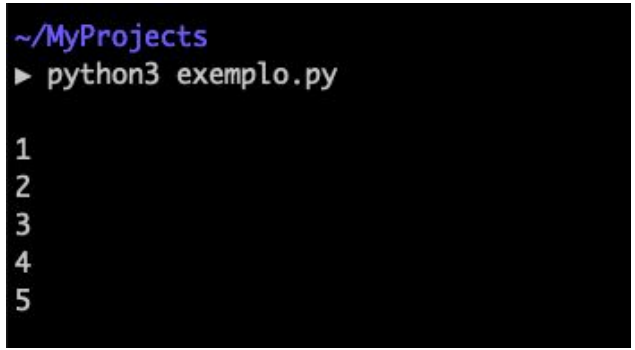
Loopings

exemplo_for.py

```
1 minha_lista = [1, 2, 3, 4, 5]
2
3 for item in minha_lista:
4     print(item)
5
```

exemplo_while.py

```
1 contador = 1
2 while contador <= 5:
3     print(contador)
4     contador += 1
5
```



Funções

Função sem parâmetros

```
1 def minha_funcao():
2     print("batata")
3     print("quente\n")
4
5 minha_funcao()
6 minha_funcao()
7
```

```
~/MyProjects
▶ python3 exemplo.py

batata
quente

batata
quente
```

Função com parâmetros

```
1 def soma(x, y):
2     return x + y
3
4 variavel = soma(3, 4)
5 print("Resultado: ", variavel)
6
```

```
~/MyProjects
▶ python3 exemplo.py

Resultado:  7
```

Funções [algumas variações]

Função tradicional

```
1 def soma(x, y):
2     return x + y
3
4 variavel = soma(3, 4)
5
```

Parâmetro com valor default

```
1 def repete(texto, qtd=7):
2     return texto * qtd
3
4 texto_repetido = repete("ola")
5 texto_repetido = repete("ola", 3)
6
```

Parâmetros tipados

```
1 def soma(x: int, y: int):
2     return x + y
3
4 variavel = soma(x=3, y=4)
5
```

Especificando tipo de retorno

```
1 def repete(texto: str, qtd: int = 7) -> str:
2     return texto * qtd
3
4 texto_repetido = repete("ola")
5
```

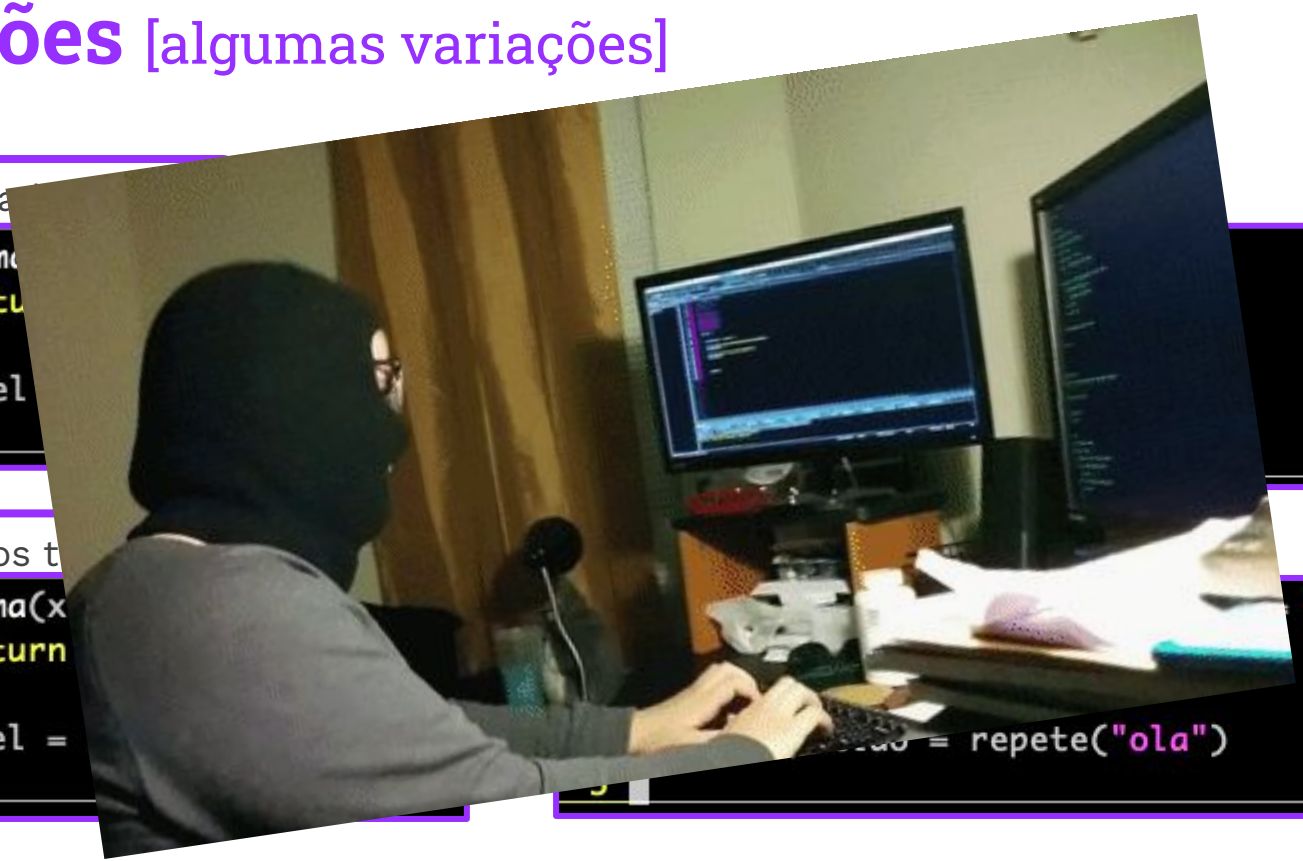
Funções [algumas variações]

Função tra

```
1 def som
2     retu
3
4 variavel
5
```

Parâmetros t

```
1 def soma(x
2     return
3
4 variavel =
5
```



```
7) -> str:

= repete("ola")
```

Classes

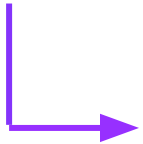
exemplo.py

```python
class Pessoa:
    qtd_ossos = 206

    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        print(f"Olá, sou {self.nome} e tenho {self.idade} anos.")
        print(f"Curiosidade: tenho {self.qtd_ossos} ossos")

eu_mesma = Pessoa(nome="Débora", idade=23)
eu_mesma.apresentar()
```



```
~/MyProjects
▶ python3 exemplo.py

Olá, sou Débora e tenho 23 anos.
Curiosidade: tenho 206 ossos
```

actions.py

```python
# This files contains your custom actions which can be used to run
# custom Python code.
#
# See this guide on how to implement these action:
# https://rasa.com/docs/rasa/core/actions/#custom-actions/

from typing import Any, Text, Dict, List

from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet


class ActionTeste(Action):
    def name(self) -> Text:
        return "action_teste"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        try:
            dispatcher.utter_message("Mensagem enviada por uma custom action.")
        except ValueError:
            dispatcher.utter_message(ValueError)
        return []

```

Cursos gratuitos de Python

Python.org | Lista de cursos de Python para quem está começando

Udemy | Python para iniciantes

Tarefa da semana

Criar uma **action** no seu chatbot RASA e colocar ela como resposta de uma intent.

- Essa intent pode ser uma nova intent ou uma já existente
- Essa action pode fazer **qualquer coisa** que você quiser, mas ela pode ser bem simples também e, por exemplo, só ser usada para **mandar uma mensagem para o usuário**.

Abre um PR com seu código no repositório do boillerplate pra gente dar uma olhada 💜

Licença

Obrigada!

Continuem brilhando!!

boss

https://github.com/orgs/BOSS-BigOpenSourceSister