



# Slots e Custom Actions

Criando custom actions com slots e acessando API



# Quem somos nós?




Bruna M.



Naiara



# Agenda

- SLOTS
    - O que são? O que comem? De onde vieram?
    - Tipos de slots
  - Bater numa api legal usando Custom Action
    - Requests
    - Python
  - Tarefa da semana
    - Criar custom action com integração com api e dar resposta pro bot
- 

# SLOTS

O que são? Do que se alimentam?






# O que é?

Quantos graus está  
fazendo agora?

... Aqui onde?

Por exemplo, se você perguntar  
pro bot **quantos graus fazem**, é  
importante que ele saiba o **onde**.






# O que é?

Quantos graus está  
fazendo agora?

... Aqui onde?

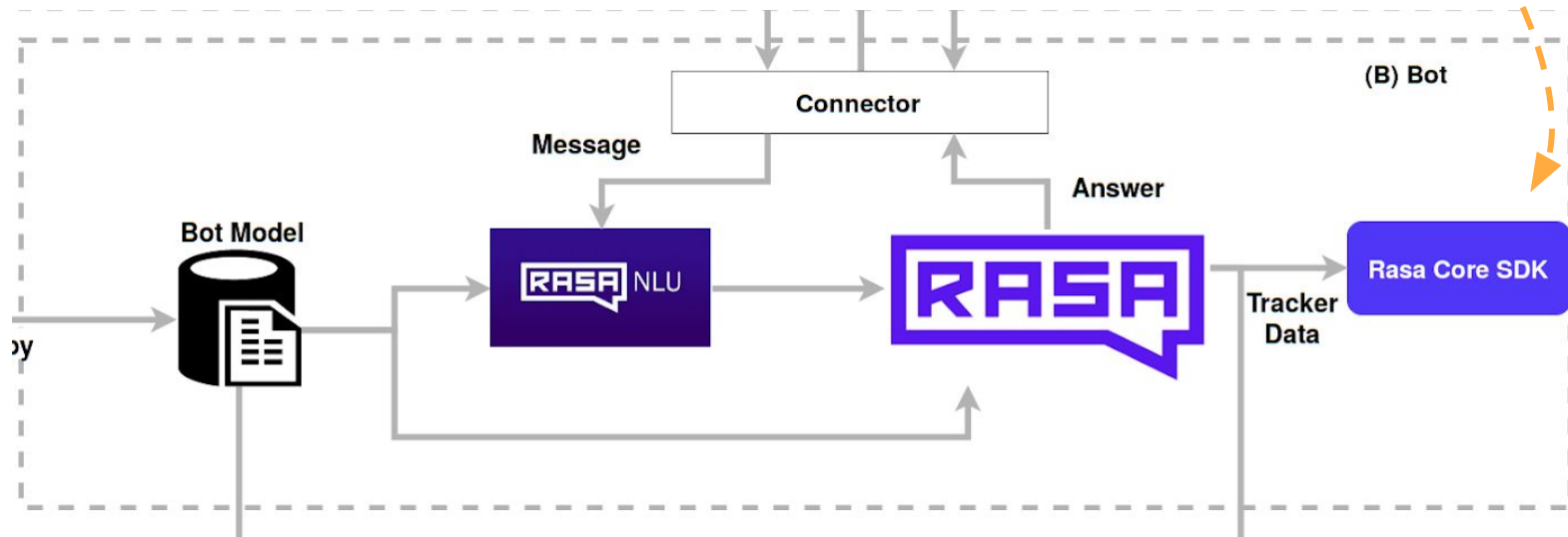
Por exemplo, se você perguntar pro bot **quantos graus fazem**, é importante que ele saiba o **onde**.

Então você utiliza um **slot** do tipo **text** chamado **cidade** para a partir daí poder checar os graus da **cidade**.



# O que é

- Esse slot apenas diz ao Rasa Core se o **text** tem algum valor.



# O que é Slot?

A **memória** do bot, que pode guardar em variáveis informações.

Atuam como armazenamento de valor-chave para informações fornecidas pelo usuário ou de fontes externas como API. O valor específico do slot não faz diferença.





# Exemplos

- Nome da pessoa com quem está falando para uma conversa personalizada.
- Retornar o valor do saldo de uma conta a partir do número da conta e senha.
- Trazer informações de um país a partir do país informado.





# Tipos de slots

- **Text** - Preferências do usuários importa se foram ou especificadas ou não, ou seja, se tem algum valor ou não.

```
slots:  
  cpf:  
    type: text
```

- Se tem algum valor o slot retorna 1, caso não tenha o slot retorna 0
- 



# Tipos de slots

- **Boolean** - Usar quando for **verdadeiro** ou **falso**

```
slots:  
  autenticado:  
    type: bool
```


- Se tem algum valor o slot retorna **true** ou **1**, caso não tenha o slot retorna **false** ou **0**
- 



# Tipos de slots

- **Categorical** - Usar quando os slots puderem ter 1 entre N valores

```
slots:  
  nivel_de_risco:  
    type: categorical  
    values:  
      - baixo  
      - medio  
      - alto
```


- Um valor padrão `__other__` automaticamente aos valores definidos pelo usuário. E todos os valores que não forem de alguma categoria já explícita são mapeados em `__other__`
- 



# Tipos de slots

- **Float** - Usar para valores contínuos

```
slots:  
  temperatura:  
    type: float  
    valor_min: -100.0  
    valor_max: 100.0
```

- Por padrão `max_value=1.0` e `min_value=0.0`. Todos os valores abaixo de `min_value` serão tratados como `min_value`, o mesmo acontece com valores acima de `max_value`.
- 




# Tipos de slots

- **List** - Lista de valores

```
slots:  
  lista_de_compra:  
    type: list
```

- O slot é definido como 1 se a lista tiver algum valor, ou seja, se não estiver vazia. E 0 caso a lista estiver vazia.

**Observação:** o comprimento da lista armazenada no slot não influencia o diálogo





# Tipos de slots

- **Unfeaturized** - Armazenar dados que não devem influenciar o fluxo do diálogo

```
slots:  
  id_usuario_interno:  
    type: unfeaturized
```

- Não tem caracterização esse slot, logo seu valor não influencia o diálogo
- 

# Slot “nome”

Aqui a gente quer guardar em um slot o nome que a pessoa informou.

Observe que temos um exemplo de frase com 4 palavras e o bot guardou o **nome** corretamente.

Para isso temos colocar nas *intents* os exemplos para que o bot entenda qual palavra é o nome.



Meu nome é Ana



Olá Ana!






nlu.md

```
10 ## intent:informar_nome
11 - O meu nome é [Ana](nome)
12 - meu nome é [Luzia](nome)
13 - Me chamo [Maria Carolina](nome)
14 - me chamo [João](nome)
15 - Pode me chamar de [Amanda](nome)
16 - Se refira a mim como [Marianna](nome)
17 - me chamam de [bruna](nome)
```

Aqui no arquivo nlu colocamos os exemplos de interação das pessoas, e podemos ver que falamos para o bot que [Ana], [Luzia], [bruna] são nomes. Por isso ele consegue salvar no slot o nome correto.



domain.yml

```
1 intents:
2   - informar_nome
3 entities:
4   - nome
5 slots:
6   nome:
7     type: text
8 responses:
9   utter_informar_nome:
10    - text: "Legal {nome}!"
```

Na domain a gente acrescenta a *intent* **informar\_nome**, a entidade **nome**, o slot com o tipo do slot para alocar a informação de **nome** e a utter de resposta quando o usuário informar seu nome

stories.md

```
7 ## informar_nome  
8 * cumprimentar  
9     - utter_cumprimentar  
10 * informar_nome  
11     - utter_informar_nome
```

Nos stories definimos o fluxo de conversa do bot para quando a pessoa informar o nome



# Custom Actions



# Tipos de actions

Actions

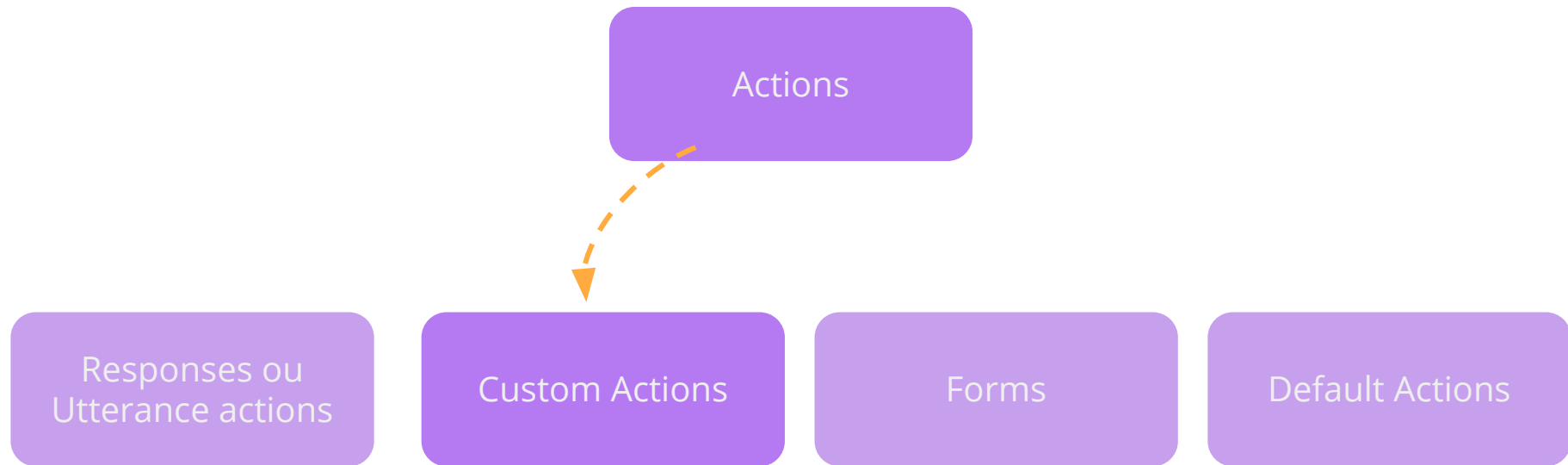
Responses ou  
Utterance actions

Custom Actions

Forms

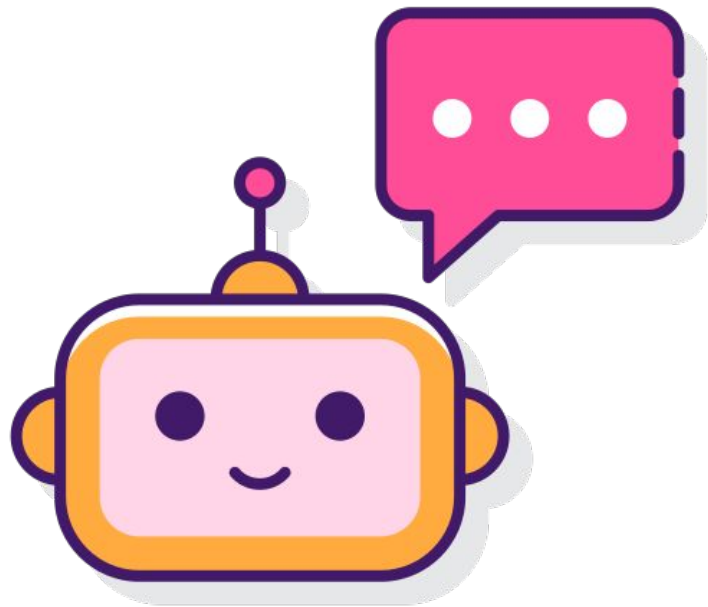
Default Actions

# Tipos de actions



# Custom Actions

Com as ações personalizadas ou custom actions você pode criar qualquer coisa! Já imaginou? Seu chatbot contando piada, dando informações sobre países, contando a última trending topic do Twitter?



# Custom Actions

Definir qual vai ser sua action, o que o bot vai fazer?

Qual informação quero do usuário? Vou utilizar que tipo de slot?

Meu bot precisa acessar alguma fonte externa, API?

Meu nome é Ana

Olá Ana!

Queria um conselho

Ana, olha que conselho legal! Não pare para pensar, pense andando.



# Custom Actions

O bot vai dar um conselho para o usuário.

Acessando uma API.

Se o usuário tiver informado o nome, salvaremos em um slot do tipo **text** e teremos duas opções de resposta para conselho: com o nome da pessoa ou sem.



Meu nome é Ana

Olá Ana!

Queria um conselho

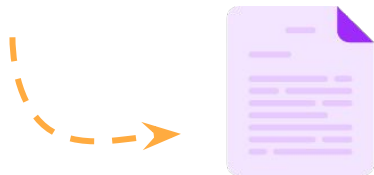
Ana, olha que conselho legal! Não pare para pensar, pense andando.



# Custom Actions

Vamos começar criando as intents:

- Uma para salvar o nome da pessoa em um slot do tipo text quando ela informar
- E outra para quando o usuário pedir um conselho



data/nlu

Meu nome é Ana

Olá Ana!

Queria um conselho

Ana, olha que conselho legal! Não pare para pensar, pense andando.

```
1 ## intent:pedir_conselho
2 - Me diga um conselho
3 - Me de um conselho
4 - Me dê um conselho
5 - queria um conselho
6 - qual conselho você me dá
7 - qual conselho você me da
8 - conselho
9
10 ## intent:informar_nome
11 - O meu nome é [Ana](nome)
12 - meu nome é [Luzia](nome)
13 - Me chamo [Maria Carolina](nome)
14 - me chamo [João](nome)
15 - Pode me chamar de [Amanda](nome)
16 - Se refira a mim como [Marianna](nome)
17 - me chamam de [bruna](nome)
```

Aqui temos as definições das *intents* de pedir conselho, que são as intenções, do que o usuário quer falar e como ele fala;

E da *intent* de informar nome, como o usuário vai falar e como identificamos o slot nome.

# Custom Actions

Criamos a Action no arquivo actions:

- Acessando a API de conselho e retornando ao final da ação
- Pegar o slot nome caso tenha e diferenciar as respostas



bot/actions/

Meu nome é Ana

Olá Ana!

Queria um conselho

Ana, olha que conselho legal! Não pare para pensar, pense andando.

# Acessando um API usando requests

Vamos usar uma API que retorna conselhos. Aqui testamos no terminal o acesso a API, e o que retorna. [Link da API](#)

```
$ python
>>> import requests
>>> req = requests.get("https://api.adviceslip.com/advice")
>>> req
<Response [200]>
>>> req.json()
{'u'slip': {'u'advice': u'Try making a list.', u'id': 216}}
>>> req.json()["slip"]["json"]
u'Try making a list.'
```

actions.py

```
1 import requests
2
3 class ActionAdvices(Action):
4     def name(self) → Text:
5         return "action_pedir_conselho"
6
7     def run(self, dispatcher, tracker, domain):
8
9         nome = tracker.get_slot('nome')
10
11         req = requests.request('GET', "https://api.adviceslip.com/advice")
12         conselho = req.json()["slip"]["advice"]
13
14         try:
15             if nome:
16                 dispatcher.utter_message("{} olha que conselho legal: {}".format(nome, conselho))
17             else:
18                 dispatcher.utter_message("Olha que conselho legal: {}".format(conselho))
19         except ValueError:
20             dispatcher.utter_message(ValueError)
```

Pega o nome do slot se  
tiver sido informado

Acessa a API e salva o  
conselho na variável  
**conselho**

Pega o nome do slot se  
tiver sido informado

# Custom Actions

Na domain:

- Inserimos as intents
- Inserimos a entidade **nome**
- Adicionamos o slot **nome** e o tipo de slot
- Inserimos a utter de resposta quando o usuário fornecer o nome
- Inserimos a action



domain

Meu nome é Ana

Olá Ana!

Queria um conselho

Ana, olha que conselho legal! Não pare para pensar, pense andando.

domain.yml

```
1 intents:
2 - pedir_conselho
3 - informar_nome
4 entities:
5 - nome
6 slots:
7   nome:
8     type: text
9 responses:
10 utter_informar_nome:
11   - text: "Legal {nome}!"
12 actions:
13 - action_pedir_conselho
```

Adicionamos a intent  
de pedir conselho

Adicionamos o nome  
da nossa action em  
**actions**



# Custom Actions

Nos stories inserimos os dois fluxos de conversas:

- Um sobre informar o nome
- E outra sobre pedir conselho



data/stories

Meu nome é Ana

Olá Ana!

Queria um conselho

Ana, olha que conselho legal! Não pare para pensar, pense andando.

stories.md

```
1 ## pedir_conselho
2 * cumprimentar
3   - utter_cumprimentar
4 * pedir_conselho
5   - action_pedir_conselho
6
7 ## informar_nome
8 * cumprimentar
9   - utter_cumprimentar
10 * informar_nome
11   - utter_informar_nome
```

Aqui adicionamos o fluxo de diálogo da *intent* de pedir conselho





# E agora?


Vamos ver se está tudo funcionando?

É sempre importante ir treinando e vendo como o bot vai respondendo.

Agora vamos treinar e testar.

**Observação:** quando alterar o arquivo actions, é importante subir o docker novamente.

```
$ make train  
$ make run-shell
```



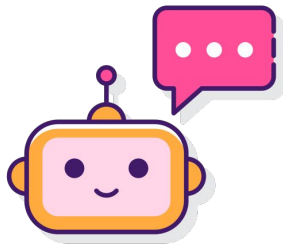
E agora?



# Docker

**Observação:** quando alterar o arquivo actions, é importante subir o docker novamente.

```
$ sudo docker ps  
$ sudo docker stop #codigo
```



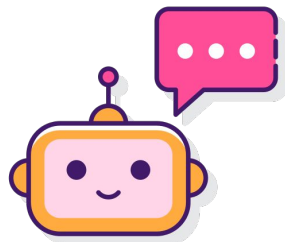
bot

O bot é diferente do  
actions



actions

# Docker



bot

Pede pra rodar



Dispatcher, retorno



actions

Quem acessa a API  
externa é o actions

```
nome = tracker.get_slot('nome')
```

```
dispatcher.utter_message("O seu telefone é {}".format(telefone))
```

```
return [SlotSet("telefone", telefone)]
```

# Arquivos trabalhados



domain

Lista de intents  
Lista de entities  
Definição de utters  
Lista de actions  
**Lista de slots**



data/nlu

Definição de intents



data/stories

Definição de stories



bot/actions/

**Definição de  
actions**



# Material de apoio

- [Documentação do RASA sobre Slots](#)
- [Documentação Requests](#)
- [Chatbot Rasa conectado com a API do Google](#)
- [Tutorial Custom Actions - Parte 1](#)
- [Tutorial Custom Actions - Parte 2](#)





# Tarefa da semana

Criar custom action com integração com uma api e dar resposta no chatbot RASA:

Isso envolve:

- Criar uma nova **custom action** acessando uma API, você pode encontrar várias APIs legais nesse site: <https://apilist.fun/>
- Bônus: Criar uma intent utilizando slot

**OBS:** Você pode usar os links dessa apresentação para identificar onde fazer essas alterações.



# Sites de APIs

- <https://apistlist.fun/>
- <https://www.goodreads.com/api/>
- <https://developer.marvel.com/>



# Licença

Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: **Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença.**

Mais detalhes sobre essa licença em: [creativecommons.org/licenses/by-nc-sa/3.0/](https://creativecommons.org/licenses/by-nc-sa/3.0/)



# Obrigada!

O que vamos fazer essa noite?



# boss



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