Aula 2: Oficina de Programação e Robótica

- Lembram-se da aula passada? Quais as operações básicas que nosso robozinho sabia fazer?
    - **move()**
    - **turn_left()**
    - **pick_beeper()**
    - **put_beeper()**
    - **turn_off()**
- Para que servem esses comandos, mesmo?
- Na verdade, esses comandos são chamados de ***métodos***. Por quê?
    - Métodos são formados por um conjunto de comandos mais simples.
- Posso criar meus próprios métodos?
    - Óbvio que sim, através da palavra chave **def**. Exemplificar com o **vire_a_direita**.

```
def vire_a_direita():
    turn_left()
    turn_left()
    turn_left()
```

    - Não se esqueçam dos dois pontos. Posso colocar quantas instruções eu quiser em um método. Como o robô sabe que a função acabou? Pelo número de espaços.
    - Coloquem alguns comandos **move**, **turn_left** e **vire_a_direita** e rodem o robô. Não se esqueçam de desligá-lo.
    - Salientar que o programa sombreia os comandos da função **vire_a_direita**, desviando da execução normal.
- Exercício: Faça um método chamado **volte_um_passo()** em que o robô volta ao passo anterior como se ele não o tivesse executado.

- Podemos criar apelidos para as funções já feitas, como **ande = move**. Nesse caso, "**ande**" é um apelido para o método **move()**.
    - Vamos fazer apelidos para **turn_left** (**vire_a_esquerda**), **pick_beeper** (**pegue_o_beeper**), **put_beeper** (**solte_o_beeper**) e **desligue** (**turn_off**).

- Vamos supor que eu queira andar 9 passos pra frente, virar à esquerda, dar mais nove passos, virar à esquerda, dar mais nove passos, virar à esquerda e dar mais nove passos. Quero rodear o muro do mundo do robô.
    - Opção 1: Fazer um método que faça o robô andar pra frente 9 vezes.
        - Por que isso melhora minha situação? Porque só preciso escrever "move" nove vezes uma única vez.
        - Qual o problema? Eu preciso escrever nove vezes a palavra "move", não tem como fazer melhor?
    - Opção 2: Quando precisamos repetir uma tarefa por um número determinado de vezes, utilizamos o comando "**for**", que tem a seguinte forma: **for i in range(9):** (salientar que tem os dois pontos). Sendo "**i**" um contador e o número dentro dos parênteses a quantidade de vezes que o **for** será executado.

      ```
      def ande():
        for i in range(9):
          move()
      ```

        - Por que isso melhora minha situação? Porque não preciso escrever "move" nove vezes.
        - Qual o problema? Imagine que eu quero andar 5 vezes pra frente, 2 pra esquerda, 1 pra esquerda, 1 pra frente... como eu faria?
    - Opção 3: "Avisar" para o método quantas vezes eu quero repetir a quantidade de passos. Para isso, dentro dos parênteses na linha **def ande():** eu coloco o nome de um argumento. Isso significa que o método só funcionará se informarmos para ele a quantidade de passos que ele deve dar. Como fazer isso? Na linha **def ande():**, escreva dentro dos parênteses o nome do argumento do método... vamos chamá-lo de **qtd_de_passos**, de modo que teremos **def ande(qtd_de_passos):**. Na linha **for i in range(9):** vamos trocar

por **for i in range(qtd_de_passos):**. Se tentarmos rodar o robô agora obteremos um erro. O erro diz que precisamos informar a qtd_de_passos. Para isso, trocamos a linha **ande()** para **ande(4)**, por exemplo. O robô dará quatro passos pra dentro.

- Exercício idiota: Faça a função **vire_a_direita** usando **for**.
- Exercício: Construa um quadrado de beepers. Para dar beepers ao robô, use o 13º botão, que parece uma antena. Role a primeira barra até conseguir 4 beepers. Depois, escreva um programa para criar um quadrado de beepers.

- Existem algumas coisas que Reeborg pode "sentir" a respeito do seu mundo, e essas são condições que você pode testar para ver o que está acontecendo nele, como testar se existe uma parede na frente, à esquerda ou à direita do robô, se o robô está olhando pro norte, se ele está carregando algum beeper e se o robô está em cima de um beeper.

Teste	Teste ao contrário	O que ele verifica:
`front_is_clear()`	`not front_is_clear()`	Existe uma parede em frente a Reeborg?
`left_is_clear()`	`not left_is_clear()`	Existe uma parede à esquerda de Reeborg?
`right_is_clear()`	`not right_is_clear()`	Existe uma parede à direita de Reeborg?
`facing_north()`	`not facing_north()`	Reeborg está virado para o Norte?
`carries_beepers()`	`not carries_beepers()`	Reeborg está carregando *beepers*?
`on_beeper()`	`not on_beeper()`	Reeborg está sobre um *beeper*?

- Em um exercício anterior, andamos pela borda do mundo do robô. Andamos nove passos porque sabíamos o tamanho do mundo, mas e se mudássemos o tamanho do mundo? O que fizemos ainda funcionaria? Não. Vamos ver? Clique no 12º botão, do lado do muro e deslize as duas barras para termos um mundo com 5 ruas e 5 avenidas. Se rodarmos o programa, o robô bate na parede. Vamos criar um novo método chamado **ande_ate_achar_parede()**. Nela, vamos usar o comando "**while**". While em inglês significa enquanto. A ideia é que o **while** vai repetir uma ação indefinidamente enquanto uma certa condição não for falsa. Nossa função **ande_ate_achar_parede()** vai utilizar a função **front_is_clear()**, que "avisa" pro robô se existe uma parede na frente dele. Nossa função vai ficar assim:

```
def ande_ate_achar_parede():
  while front_is_clear():
    move()
```

- Vamos criar mundos com novos tamanhos e testar pra ver se funciona? Bateu na parede?

- O que acontece se quisermos pegar um beeper onde não existe? Ocorre erro.
- E como podemos evitar isso? Se existisse alguma forma de testar se estamos acima de um beeper antes de pegá-lo conseguiríamos evitar esse erro. Existe uma função chamada **on_beeper()**, que nos avisa se o robô está pisando em um beeper. Mas como podemos testar se estamos pisando em um beeper? Através do comando "**if**". If, em inglês, significa uma condição. O comando **if** no mundo do Reeborg pode se apresentar de duas formas...

1. Um comando *if simples*, para situações em que uma ação pode ou não ser executada:

   if *condição:*
   comandos a serem executados se a condição for verdadeira

2. Um comando *if-else*, para situações em que você deve escolher entre duas ações diferentes:

   if *condição:*
   comandos a serem executados se a condição for verdadeira
   else*:*
   comandos a serem executados se a condição for falsa

- Voltando aos beepers, vamos, então, fazer da seguinte forma...

```
if on_beeper():
  pick_beeper()
```

Assim, o robô só vai pegar um beeper que esteja na sua posição atual no mundo.

```
x = 2
if x == 2:
  turn_left()
else
 turn_left()
 turn_left()
 turn_left()
move()
```

Depois coloque **x = 1** e rode o programa.