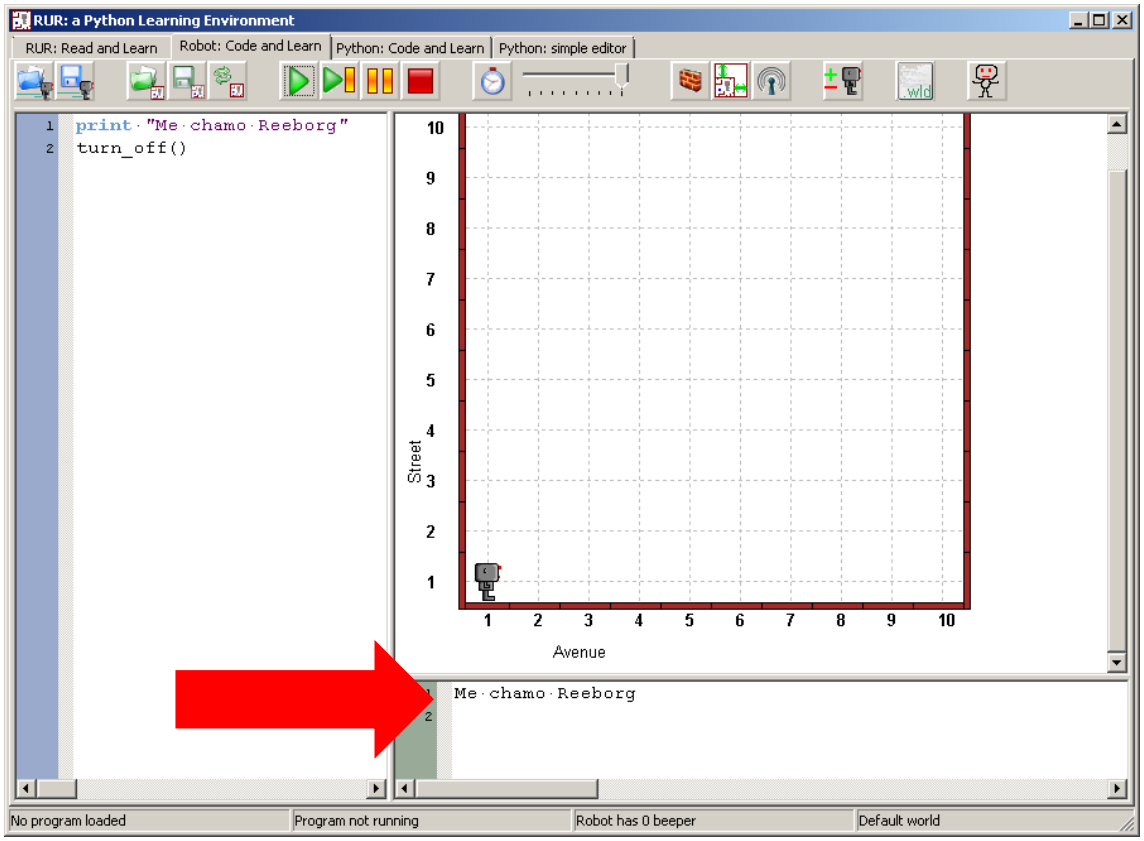
O que não contamos pra vocês nas aulas

Print

Existe uma função no Reeborg que escreve mensagens na parte inferior do programa, essa função se chama *print*. Veja o exemplo abaixo.

```
1 print "Me chamo Reeborg"
2 turn_off()
```



Operadores

A linguagem Python tem alguns operadores. Considere a = 5, b = 2.

Operadores Matemáticos

Operador	Nome	Exemplo	Valor de c
=	Atribuição	c = 1	1
+	Adição	c = 2 + b	4
-	Subtração	c = 2 – a	-3
*	Produto	c = 3 * a	15
/	Divisão	c = a / 2	2.5
%	Módulo (resto da divisão)	c = a % 2	1
**	Expoente	c = a ** 2	25
//	Divisão inteira	c = a // 2	2.0

Operadores de Comparação

Os operadores de comparação nos informam se uma determinada comparação é verdadeira ou falsa.

Operador	Nome	Exemplo	Valor
==	Igualdade	a == b	Falso
!=	Diferente	a != b	Verdadeiro
<>	Diferente	a <> b	Verdadeiro
>	Maior que	a > b	Verdadeiro
<	Menor que	a < b	Falso
>=	Maior ou igual que	a >= b	Verdadeiro
<=	Menor ou igual que	a <= b	Falso

Operadores Lógicos

São utilizados para juntar duas comparações e obter apenas um resultado lógico (verdadeiro ou falso).

Operador	Nome	Tabela verdade
and	E	O operador “and” só resultará em verdadeiro se as duas condições forem verdadeiras. Se qualquer condição for falsa, o “and” resultará em falso.
or	Ou	O operador “or” resultará verdadeiro se ao menos uma das

Operador	Nome	Tabela verdade
		condições testadas for verdadeira. Somente quando as duas condições forem falsas, o teste resultará em falso.
not	Não	O “not” muda o valor lógico de uma variável.

Desejamos fazer o Reeborg andar enquanto ele não encontrar uma parede e enquanto houver beepers para ele pegar. Nossa solução está a seguir, teste-a colocando alguns beepers por onde o robô anda, você verá que o robô só se moverá quando não tiver uma parede na frente dele **e** se ele estiver sobre um beeper.

```
while front_is_clear() and on_beeper():
    pick_beeper()
    move()
turn_off()
```

Variáveis

Variáveis são espaços na memória principal do computador que guardam algum valor momentâneo, que pode ser importante ser guardado por um espaço de tempo curto. No mundo do Reeborg, uma variável é criada apenas digitando-se seu nome no meio do código e se atribuindo algum valor a ela.

```
a = 3
```

Figura 1 Criação de uma variável de nome "a" com valor inicial 3.

Lembra-se do *for*? Quando criamos a estrutura *for i in range(3)* estamos, na verdade, criando uma variável de nome *i* que varia de 0 a 2 (porque 3 é o limite). Dessa forma, as duas construções abaixo funcionam, **exatamente**, da mesma forma.

```
for i in range(3):
    move()
turn_off()
```

```
i = 0
while i < 3:
    move()
    i = i + 1
turn_off()
```

Figura 2 Movendo o robô três vezes para frente, utilizando *for* (construção da esquerda) e *while* (direita). Os dois códigos são equivalente e funcionam da mesma forma.

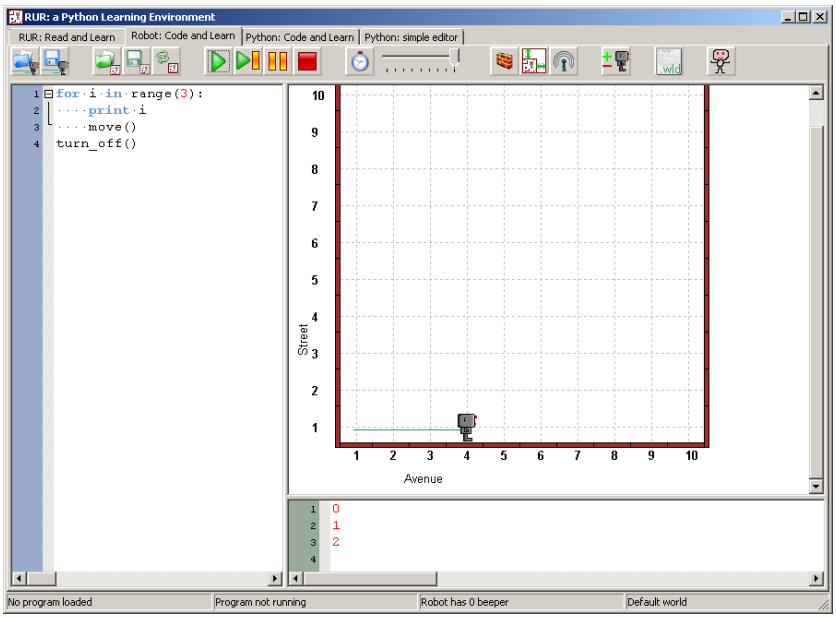
Os nomes das variáveis devem seguir duas regras.

1. Só podem conter letras, números e sublinhado (_);

2. Não pode começar por número.

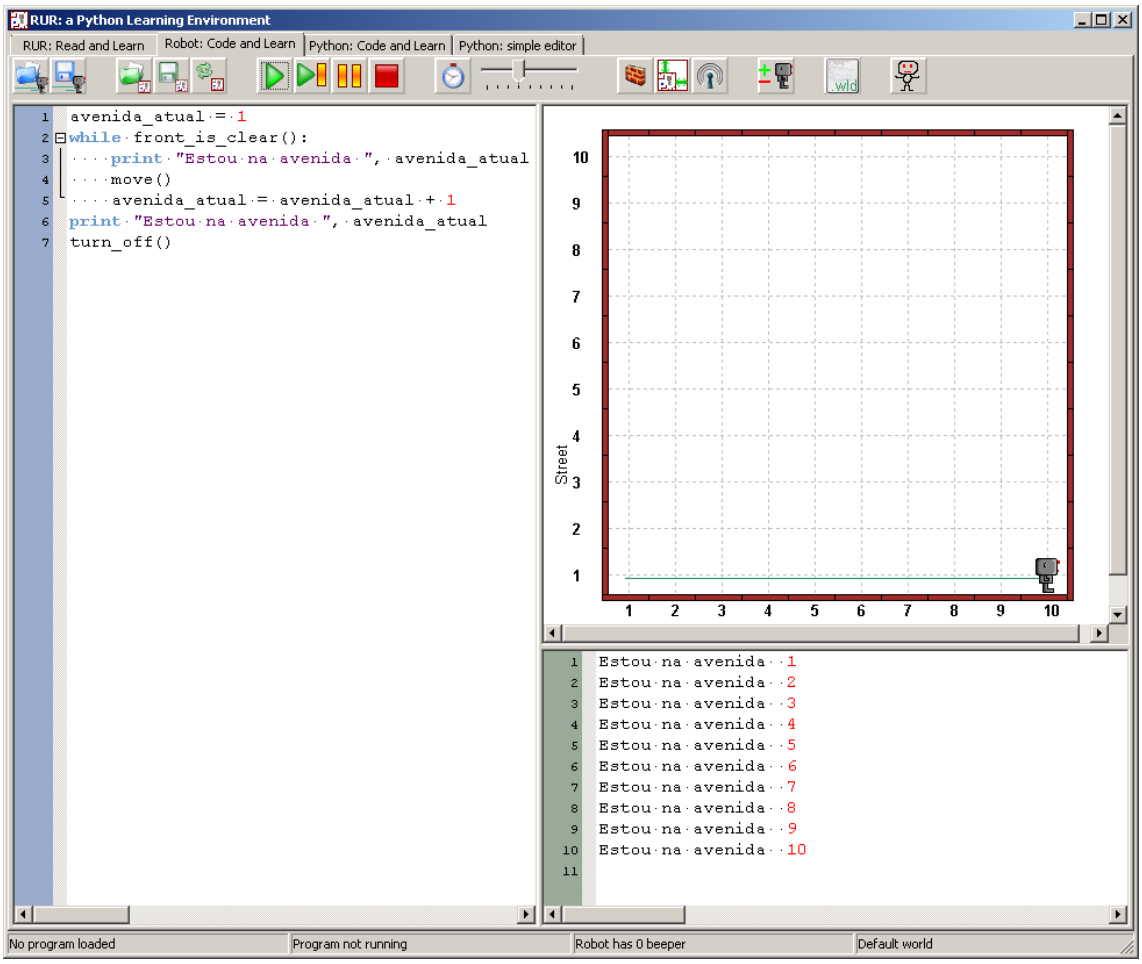
É possível utilizar a função *print* para exibir o estado de variáveis, veja os códigos abaixo:

```
for i in range(3):
    print i
    move()
turn_off()
```



Também é possível utilizar *print* concatenando (juntando) frases fixas e valores de variáveis. Por exemplo, para fazer com que o Reebord nos diga em que avenida ele está.

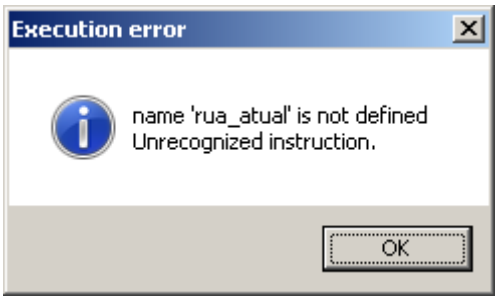
```
avenida_atual = 1
while front_is_clear():
    print "Estou na avenida ", avenida_atual
    move()
    avenida_atual = avenida_atual + 1
print "Estou na avenida ", avenida_atual
turn_off()
```



Toda variável, antes de ser lida, precisa ser iniciada. Observe o código abaixo.

```
1 if front_is_clear():
2     rua_alvo = rua_atual * 2
3 while rua_atual < rua_alvo:
4     move()
5     rua_atual = rua_atual + 1
6 turn_off()
```

A linha dois, se executada, vai gerar o erro abaixo já que escrevemos *rua_alvo = rua_atual * 2*, ou seja, colocamos o valor da variável *rua_alvo* dependente do valor da variável *rua_atual*. Como não foi atribuído nenhum valor inicial para *rua_atual*, o Reeborg simplesmente não sabe o que fazer, e gera esse erro. Para consertá-lo, basta iniciar *rua_atual* com um valor qualquer.



Além de números inteiros, variáveis podem ter números reais, textos e valores lógicos.

Execute o programa abaixo e verifique o resultado:

```
1  nome = "Reeborg"
2  peso = 4.5
3  altura = 0.45
4  imc = peso / (altura * altura)
5  print "IMC = ", imc
6  if imc >= 30:
7      print nome, " está acima do peso"
8  else:
9      print nome, " está no peso ideal"
10 turn_off()
```

Os valores lógicos que uma variável pode guardar são *True* (verdadeiro) ou *False* (falso). Valores lógicos são utilizados quando precisamos testar a veracidade de alguma variável. O programa abaixo faz o robô andar até encontrar um beeper. Para simplificar, supomos que sempre haverá um beeper antes do Reeborg encontrar uma parede.

```
1  encontrei_beeper = False
2  while not encontrei_beeper:
3      if on_beeper():
4          encontrei_beeper = True
5      else:
6          move()
7  turn_off()
```

Elif

Vimos que quando nosso robô precisa tomar uma decisão utilizamos o comando *if*. Lembram-se do exemplo abaixo? Movemos o robô e, se encontrarmos um beeper, nós o pegamos.

```
while front_is_clear():
    if on_beeper():
        pick_beeper()
    move()
turn_off()
```

Já o *else* era utilizado quando a condição no *if* fosse falsa. O programa abaixo coloca um beeper onde não houver e pega os que estiverem no meio do caminho.

```
while front_is_clear():
    if on_beeper():
        pick_beeper()
    else:
        put_beeper()
    move()
turn_off()
```

É possível perceber que o *else* não tem condição, ou seja, se a condição da linha dois for verdadeira, a linha três será executada. Se a condição for falsa, a linha cinco será executada.

Às vezes, gostaríamos de testar outra condição ao invés de automaticamente cair no *else*. Para isso, existe o comando *elif*. Você pode usar quantos comandos *elif* quiser, mas eles precisam sempre estar ligados a um *if* inicial.

Voltando ao exemplo anterior, podemos utilizar a função *carries_beepers()* já vista anteriormente para colocar um beeper de forma segura, já que um erro ocorre se chamarmos a função *put_beeper()* e o robô não estiver carregando nenhum beeper consigo.

```
while front_is_clear():
    if on_beeper():
        pick_beeper()
    elif carries_beepers():
        put_beeper()
    move()
turn_off()
```

Utilizando a função *print*, já vista, podemos levemente modificar o código acima apenas para ilustrar a cadeia *if – elif – else*.

```
while front_is_clear():
    if on_beeper():
        pick_beeper()
    elif carries_beepers():
        put_beeper()
    else:
        print "Não há beepers, mas não carrego beepers para colocar aqui"
    move()
turn_off()
```

Break

Quando estamos executando um *loop* (*for* ou *while*), podemos forçar o loop parar com a palavra reservada *break*. Embora sempre seja possível fazer um *loop* sendo controlado apenas por suas condições através de operadores lógicos, o uso do *break* pode facilitar o entendimento do código escrito, em algumas vezes. O exemplo de movimentar o robô até encontrarmos uma parede ou um beeper, o que vier primeiro, pode ser feito da seguinte forma utilizando *break*.

```
# o robô vai andar até encontrar uma parede
while front_is_clear():
    # se ele encontrar um beeper, ele pula
    # fora do while
    if on_beeper():
        break
    # note que não precisamos de um else
    # aqui. Esse move() só será executado
    # se o if não for verdadeiro, uma vez
    # que o move está dentro do while e,
    # caso seja encontrado um beeper, o
    # break força a saída do loop
    move()
turn_off()
```

Algo importante sobre o *break* é que ele só sai do *loop* mais interno. Por exemplo, considere um robô que anda até encontrar uma parede e pega, no máximo, três beepers por vez. Se em uma esquina existem cinco beepers, o robô pega três e deixa dois lá, se na próxima esquina existem mais dois beepers, ele pega os dois, e assim por diante.

```
while front_is_clear():
    if on_beeper():
        beepers_pegos = 0
        while on_beeper():
            pick_beeper()
            beepers_pegos = beepers_pegos + 1
            if beepers_pegos == 3:
                # esse break só interrompe o while
                # mais interno. O while
                # front_is_clear continua sendo
                # executado
                break
    move()
turn_off()
```

Continue

Utilizamos o *break* quando desejamos sair de um loop. O *continue* apenas ignora a iteração atual do loop e passa pra próxima. Vamos fazer um robô que só coloca os beepers em avenidas pares.

```
def coloca_beeper_seguro():
    if carries_beepers():
        put_beeper()

avenida_atual = 1
while front_is_clear():
    move()
    avenida_atual = avenida_atual + 1
    # se o resto da divisão = 0,
    # estamos em uma avenida par
    if avenida_atual % 2 == 0:
        coloca_beeper_seguro()
turn_off()
```

Com *continue*, o programa acima poderia ser escrito da seguinte forma.

```
def coloca_beeper_seguro():
    if carries_beepers():
        put_beeper()

avenida_atual = 1
while front_is_clear():
    move()
    avenida_atual = avenida_atual + 1
    # se o resto da divisão for
    # diferente de zero, estamos em uma
    # avenida ímpar
    if avenida_atual % 2 != 0:
        # como o continue passa para a
        # próxima iteração do loop, o
        # coloca_beeper_seguro não
        # será executado, caso o if
        # seja verdadeiro
        continue
    coloca_beeper_seguro()
turn_off()
```

Assim como o *break*, o uso do *continue* **sempre** pode ser evitado. Utilize-o apenas se o seu uso deixar o código mais legível. O uso indiscriminado desses dois comandos pode deixar seu código confuso e difícil de ser entendido.

Parâmetros para Métodos

Já sabemos criar métodos para o Reeborg através da palavra reservada *def*. Às vezes precisamos passar algumas informações para que os métodos funcionem.

Considere o trecho de código abaixo:

```
# Faz o robô dar várias
# voltas de 360 graus.
# A quantidade de vezes
# que o robô vai girar
# será informada pela
# variável vezes, a ser
# informada quando o
# método for chamado.
def dar_voltas(vezes):
    for i in range(vezes):
        dar_uma_volta()

# Faz o robô dar uma
# volta de 360 graus.
def dar_uma_volta():
    for i in range(4):
        turn_left()

# Chama o método dar_voltas
# informando ao método que
# o robô deve dar 10 voltas.
dar_voltas(10)
turn_off()
```

Quando desejamos passar mais de um parâmetro utilizamos a vírgula. Veja o exemplo abaixo, onde o método *mover_ate* recebe dois parâmetros (rua e avenida) e move o robô até a rua e a avenida solicitadas.

```
def turn_right():
    for i in range(3):
        turn_left()

def mover_ate(rua, avenida):
    turn_left()
    # Subtraimos um porque o
    # robô já está na rua 1
    for i in range(rua - 1):
        move()
    turn_right()
    # Subtraimos um porque o
    # robô já está na avenida 1
    for i in range(avenida - 1):
        move()

# informando o método mover_ate
# que gostaríamos de ir até a
# rua 2, avenida 4.
mover_ate(2, 4)
turn_off()
```

Utilizamos a palavra reservada *return* para sair de uma função. Veja.

```
def turn_right():
    for i in range(3):
        turn_left()

def mover_ate(rua, avenida):
    if rua == 1 and avenida == 1:
        return
    turn_left()
    for i in range(rua - 1):
        move()
    turn_right()
    for i in range(avenida - 1):
        move()

mover_ate(1, 1)
turn_off()
```

No exemplo acima, se a rua e a avenida para a qual desejamos nos mover forem iguais a 1, significa que o robô não precisa se mover. Nesse caso, saímos da função sem executá-la.

```
def turn_right():
    for i in range(3):
        turn_left()

def mover_ate(rua, avenida):
    if rua == 1 and avenida == 1:
        #Nesse caso, o robô já
        #está no seu posicionamento
        #final.
        return True
    turn_left()
    for i in range(rua - 1):
        if not front_is_clear():
            return False
        move()
    turn_right()
    for i in range(avenida - 1):
        if not front_is_clear():
            return False
        move()
    return True

if mover_ate(12, 1):
    print "Reeborg chegou corretamente"
else:
    print "Reeborg nao chegou"
turn_off()
```

Eventualmente, podemos utilizar a palavra *return* seguida de algum valor para indicar algo para quem chamou o método. Por exemplo, no exemplo acima, o que aconteceria se mandássemos o robô se mover até a rua 12 em um mundo com apenas 10 ruas? Ele não se moveria corretamente. Nesse caso, podemos utilizar os valores lógicos *True* ou *False* para indicar se o robô se movimentou corretamente. Veja ao lado, qual *print* é executado?

Pode-se retornar qualquer tipo de informação através de *return*, o que o programa abaixo faz?

```
def conta_beepers_esquina():
    total_beepers = 0
    while on_beeper():
        pick_beeper()
        total_beepers = total_beepers + 1
    for i in range(total_beepers):
        put_beeper()
    return total_beepers


total_beepers_rua = 0
while front_is_clear():
    total_beepers_rua = total_beepers_rua + conta_beepers_esquina()
    move()
print "Temos ", total_beepers_rua, " beepers nessa rua"
turn_off()
```

Qualquer dúvida não deixem de contatar os professores da disciplina!!!