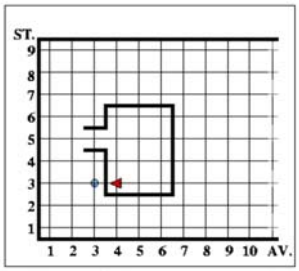
TASK: Every morning Reeborg is awakened in bed when his newspaper, represented by a beeper, is thrown onto the front porch of his house. Program Reeborg to retrieve his paper and bring it back to bed with him. The figure below illustrates the initial situation. (Newspaper is at street 3, avenue 3.) The final situation must have Reeborg back in bed (same corner, same direction) with the newspaper (that is, in his pocket).



Initial situation for the Newspaper Retrieval Task

Extending *Reeborg's* Vocabulary

Why should we extend *Reeborg's* vocabulary?

- ***Reeborg's* vocabulary is limited.**
- **For example, *Reeborg* does not understand a turn_right() command.**

  He can only turn right by executing three turn_left() commands.

- **As a second example, suppose we need to program *Reeborg* to travel over vast distances.**
- **Assume that the robot must move east ten miles (a mile is eight blocks long), pick up a beeper, and then move another ten miles north.**
- ***Reeborg* understands how to move a block, but not a mile.**
- **Conversion from miles to blocks is straightforward, but results in a very long and unreadable program. How many move() instructions would be included?**

Reeborg can "learn" new commands

- **Our programs can furnish him with a *dictionary* of useful instruction names and their definitions.**
- **Given this ability to extend *Reeborg's* vocabulary, we can solve our problems using instructions more natural to our way of thinking.**

Defining a new function

```
def identifier():
    statement(s) that define the function
    . . .
    . . .
```

- **Using this function definition mechanism to create a kind of dictionary for *Reeborg*, new commands can be defined to extend *Reeborg's* vocabulary by writing functions.**
- **For example, although *Reeborg* does not have a predefined turn_right() command, we can user-define this instruction by writing the function shown on the next slide.**

```
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

- **Similarly, we can define move_a_mile() to mean eight move() instructions.**
- **Thus instead of 160 move() instructions we would need only 20 move_a_mile() instructions.**

```
def move_a_mile():
    move()
    move()
    move()
    move()
    move()
    move()
    move()
    move()
```

Benefits of extending *Reeborg's vocabulary*

- **We can use instructions "natural" to us.**
- **As in the 20 mile problem, the size of our programs can be significantly reduced.**
- **The smaller program is much easier to read and understand.**

```
# Comments describing
# what program does.

move()
pick_beeper()
move()
turn_left()
move()
move()
put_beeper()
move()
```

```
turn_left()
turn_left()
turn_left()
move()
turn_left()
turn_left()
turn_left()
move()
turn_left()
turn_left()
turn_left()
turn_off()
```

```
# Comments describing
# what program does.

move()
pick_beeper()
move()
turn_left()
move()
move()
put_beeper()
move()
```

```
turn_left()
turn_left()
turn_left()      } turn_right()
move()
turn_left()
turn_left()      } turn_right()
turn_left()
move()
turn_left()
turn_left()      } turn_right()
turn_left()
turn_off()
```

```
# Comments describing
# what program does.

def turn_right():
    turn_left()
    turn_left()
    turn_left()

move()
pick_beeper()
move()
turn_left()
move()
move()
put_beeper()
move()
```

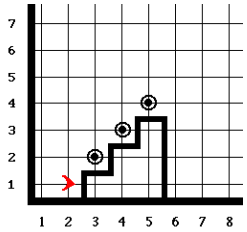
```
turn_right()

move()

turn_right()

move()

turn_right()

turn_off()
```

```
# Comments describing
# what program does.

def turn_right():
    turn_left()
    turn_left()
    turn_left()

move()
pick_beeper()
move()
turn_left()
move()
move()
put_beeper()
move()
```

```
turn_right()
move()
turn_right()
move()
turn_right()
turn_off()
```

A Stair Cleaning Task

- ***Reeborg*** **is supposed to climb stairs and pick up the beepers on each step. When he is done, he should be standing on the top step, facing East.**



If we invent a new instruction called climb_a_step() then the program can be written as:

```
climb_a_step()
pick_beeper()
climb_a_step()
pick_beeper()
climb_a_step()
pick_beeper()
turn_off()
```

Now we must write the function climb_a_step()

```
def climb_a_step():
    turn_left()
    move()
    turn_right()
    move()
```

Notice that this function depends on turn_right() , an instruction that we wrote previously.

So that we can place our "main" code at the beginning of our program, we enclose that code in a main() function that we then invoke at the bottom of our program.

```
def main():
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    turn_off()
```

So that we can place our "main" code at the beginning of our program, we enclose that code in a main() function that we then invoke at the bottom of our program.

```
def main():
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    turn_off()
```

```
# A Stair Cleaning Program
def  main():
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    climb_a_step()
    pick_beeper()
    turn_off()

# Climb on to next step
def climb_a_step():
    turn_left()
    move()
    turn_right()
    move()

# Pivot Reeborg  90 degrees to right
def  turn_right():
    turn_left()
    turn_left()
    turn_left()

# Invoke main()
main()
```

Recapping how to do function definitions

- The reserved word `def` starts a new function followed by the name of the function followed by parentheses and a colon.
- The name of the function specifies what the procedure is intended to do.
- The statements that perform the task are listed after the function header indented relative to the header. These statements specify how the new instruction does what the name implies.
- The two must match exactly – otherwise one or both need to be changed.

- To demonstrate this, there are no restrictions prohibiting the following instructions definition:

```
def turn_right():
    turn_left()
    turn_left()
```

- This is a perfectly legal definition, but it is wrong because it does not accomplish what it should.