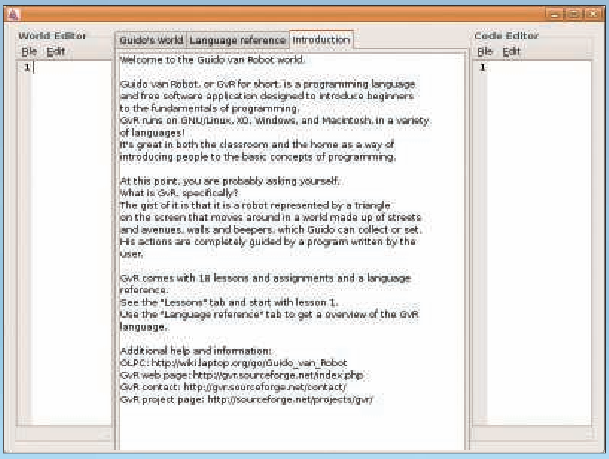
An up-to-date look at free software and its makers

PROJECTS ON THE MOVE

Python inventor Guido van Rossum is known to have a good sense of humor and probably had a good laugh when he heard about the Guido van Robot project. If you are learning how to program, or would like to teach others, this software gives you a light-hearted introduction. **BY CARSTEN SCHNOBER**

Perhaps you've heard of Python creator Guido van Rossum, but you might not be familiar with Karel the Robot. Almost 30 years ago, Richard Pattis invented a robot called Karel and presented him to the general public in his 1981 book *Karel the Robot: A Gentle Introduction for the Art of Programming* [1]. This virtual character has the advantage of living in a simple world in which his biggest challenges are maneuvering around obstacles – namely, walls – and picking up beepers. To tackle his tasks, Karel can be programmed in a simple language with an understandable instruction set and without variables.

The original Karel the Robot programming language was based on Pascal, but many variants have been invented since then to give learners a simpler approach to other programming languages. One of these variants is Guido van Robot, a character in the long-standing tradition of simulated, programmable robots, whose task it is to introduce newcomers to Python [2] (Figure 1). As a tribute to Python's Dutch inventor, Guido van Rossum [3], the students who programmed the Python teaching language in 2001 called it Guido van Robot [4]. Version 3.3 is now available.



```
Welcome to the Guido van Robot world.

Guido van Robot, or GvR for short, is a programming language
and free software application designed to introduce beginners
to the fundamentals of programming.
GvR runs on GNU/Linux, XO, Windows, and Macintosh, in a variety
of languages!
It's great in both the classroom and the home as a way of
introducing people to the basic concepts of programming.

At this point, you are probably asking yourself,
What is GvR, specifically?
The gist of it is that it is a robot represented by a triangle
on the screen that moves around in a world made up of streets
and avenues, walls and beepers, which Guido can collect or set.
His actions are completely guided by a program written by the
user.

GvR comes with 18 lessons and assignments and a language
reference.
See the "Lessons" tab and start with lesson 1.
Use the "Language reference" tab to get a overview of the GvR
language.

Additional help and information:
OLPC: http://wiki.laptop.org/go/Guido_van_Robot
GvR web page: http://gvr.sourceforge.net/index.php
GvR contact: http://gvr.sourceforge.net/contact/
GvR project page: http://sourceforge.net/projects/gvr/
```

Figure 1: Guido van Robot makes the world of programming more easily accessible by providing an interface and a language of its own, based on Python.

Guido – the robot that is, not the Python inventor – can understand five commands: *move*, *turnleft*, *pickbeeper*, *putbeeper*, and *turnoff*. The first two commands tell the robot to move or turn left; *pick* and *put* tell it to pick up and drop beepers, and *turnoff* quits the program (Figure 2).

Besides the imperatives listed here, Guido – like most other programming languages – gives users loops and conditions. The latter are introduced by the key words *if*, *elif*, and *else*; the *while* key word supports conditional execution. The *do X* command tells Guido to repeat the following command block a certain number of times.

View of the World

To be able to react to its environment, Guido van Robot uses 18 test instructions to explore its world, including *front_is_clear*, *left_is_clear*, *right_is_clear*, and *front_is_blocked*, that define the existence of walls. The objects lying around on the playing field are referred

orov, Fotolia

to as beepers. The instructions *next_to_a_beeper* and *any_beepers_in_beeper_bag* and corresponding negatives tell Guido where the beepers are.

The last remaining test category contains the *facing_north* and *not_facing_north* commands and variants, which use the other three directions instead of *north*. Typical Guido van Robot instructions look like:

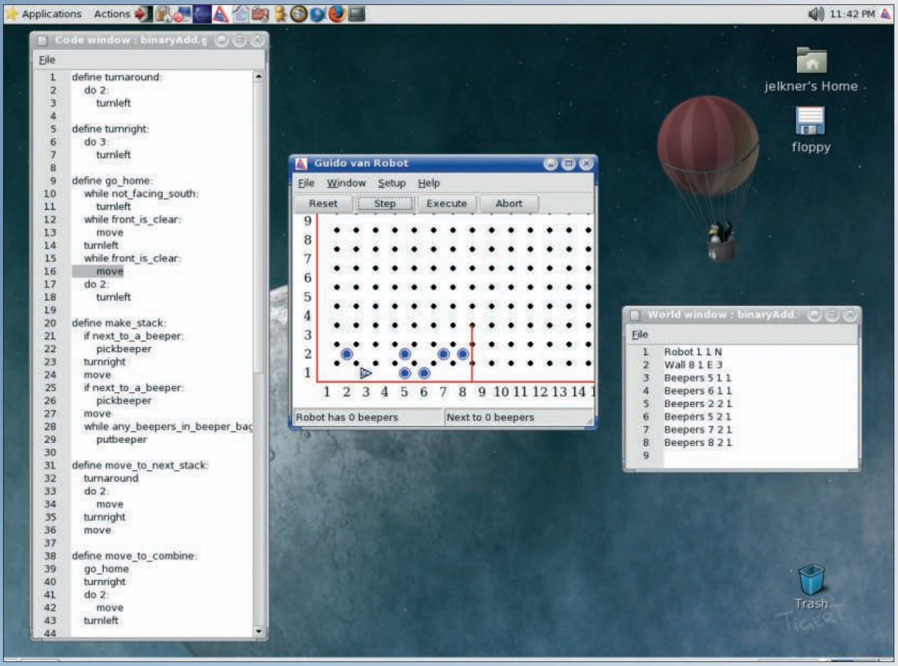
```
if next_to_a_beeper:
  pickbeeper
elif front_is_clear
:
  move
```

What makes Guido van Robot so attractive to newcomers is that the environment has its own interface, comprising the *Code Editor*, *World Editor*, and *Guido's World* area. Python newcomers can watch their robot tour the world while their program is running.

Define-It-Yourself

If you are not happy with the existing instruction set, you can define your own instructions based on it. Put the *define Name* keyword at the start of a definition block, which can contain simple instructions, loops, and conditions. Here's an example for a step backward:

```
define back:
  do 2:
    turnleft
  move
  do 2:
    turnleft
```



```
define turnaround:
  do 2:
    turnleft

define turnright:
  do 3:
    turnleft

define go_home:
  while not_facing_south:
    turnleft
  while front_is_clear:
    move
  turnleft
  while front_is_clear:
    move
  do 2:
    turnleft

define make_stack:
  if next_to_a_beeper:
    pickbeeper
  turnright
  move
  if next_to_a_beeper:
    pickbeeper
  move
  while any_beepers_in_beeper_bag
    putbeeper

define move_to_next_stack:
  turnaround
  do 2:
    move
  turnright
  move

define move_to_combine:
  go_home
  turnright
  do 2:
    move
  turnleft
```

```
Robot has 0 beepers    Next to 0 beepers
```

```
Robot 1 1 N
Wall 8 1 E 3
Beepers 5 1 1
Beepers 6 1 1
Beepers 2 2 1
Beepers 5 2 1
Beepers 7 2 1
Beepers 8 2 1
```

Figure 2: The Guido van Robot programming language running on Linux. The robot understands five commands: move, turnleft, pickbeeper, putbeeper, and turnoff.

What Guido van Robot now lacks is an environment through which the program can guide him.

Fortunately, the software package includes three worlds: *boring*, *escape1*, and *maze1*. The first lacks barriers, which is a good thing in normal circumstances, but here, its lack of events makes it unsuitable for testing algorithms. *escape1* is a more interesting prospect; a world that comprises a room with an exit. Trying to get to the exit as quickly as possible is a more useful programming task.

The third environment is a small maze, as you might have guessed from the name, *maze1*, in which Guido finds the first beeper.

If you want to set your robot on more ambitious tasks, you easily can define your own worlds. For this, the World Editor understands the *WALL* command.

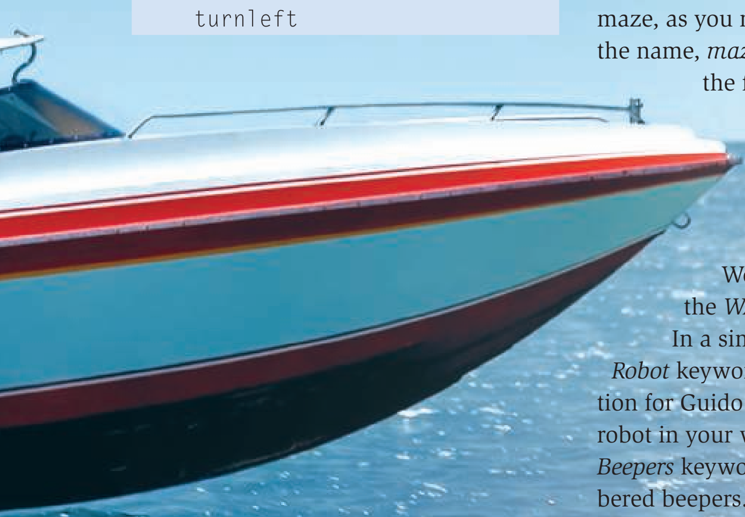
In a similar way, you can use the *Robot* keyword to set the starting position for Guido van Robot. To feed the robot in your world, you need to use the *Beepers* keyword to lay a trail of numbered beepers.

The first two arguments define the position of the object and are followed by an arbitrary number that the program uses to label the beeper.

After designing a world and writing a robot program, you can press the *Execute* button to launch the robot. The code editor highlights the current line, making it easy to follow loops and conditions. If everything is happening too fast for you to follow, you can press the *Step* button to step through the instructions one by one.

If you are familiar with Python or any other programming language, you will easily master programming Guido van Robot and its numerous relatives. Programming the robot can be fun even for serious programmers. The effortless way in which you can maneuver the robot through small worlds is a rewarding experience.

Of course, Guido van Robot is suitable for teaching, which is what the project is really about (Figure 2). If you never have tried programming, this gives you a fun approach to the buzz language Python and could be your ticket to entering the wacky world of coding. ■



INFO

[1] *Karel the Robot*: *http://www.mtsu.edu/~untch/karel/book.html*

[2] Python: *http://www.python.org*

[3] Guido van Rossum: *http://www.python.org/~guido*

[4] Guido van Robot: *http://gvr.sourceforge.net*