4: Higher! Lower!

Gareth McCaughan Revision 1.8, May 14, 2001

Credits

For the LaTeX source of this sheet, and for more information on LiveWires and on this course, see the LiveWires web site at http://www.livewires.org.uk/python/

Introduction

In this sheet, we'll write a simple guessing game: the computer (or, another player) picks a number, and you have to guess it. You get told after each guess whether the guess was too high or too low.

What you need to know

To do this sheet, you need to know:

- How to edit and run a Python program: see Sheet R (*Running Python*)
- The basic bits of Python, from sheets 1 and 2

Planning it out

A typical game might go something like this:

```
OK, I've thought of a number between 1 and 1000.

Make a guess: 200
That's too low.

Make a guess: 500
That's too high.

Make a guess: 345
That's too high.

Make a guess: 300
That was my number. Well done!

You took 4 guesses.
Would you like another game? no
OK. Bye!
```

So, the program needs to do the following things.

- Choose a number somehow, and tell the player that it's done so.
- Repeatedly:
  - Ask the player for a number
  - Report on whether it's too high, too low or just right

  (Do this until the player guesses the answer.)
- Keep track of the number of guesses the player made.
- Finally, report on the player's performance and offer the chance of another game.
- If they want another game, start over again.

Easy stuff

The first thing on that list, you already know how to do from Sheet 2. So: write a program that chooses a random number between 1 and 1000, puts it in a variable, and displays a message saying that it's done so. (Don't forget to say `from livewires import *` at the start!)

Another thing that you should already know how to do is asking the player for a number and saying whether their number is too high, too low, or just right. You've already done something very like this for Sheet 2. You'll need to use `if`.

So: add some lines to your program so that after picking a number and telling you it's picked a number, it asks you for a guess and tells you how you did.

Over and over and over again, again

In Sheet 2, there was a section called "Over and over and over again", about "`for` loops". Those are very useful when you know in advance how many times you want to do something, but much less useful when you don't know at the start when you'll want to end. That's our problem here: we don't know how many guesses the player will make before guessing the right answer.

For this sort of problem, Python has another kind of loop. It's called a "`while` loop", because (1) it begins with the word `while` and (2) it tells the computer to "do so-and-so *while* such-and-such is true".

Here's a simple example. Try it.

```
x=1
while x<100:
  print 'x is', x
  x = x+6
```

You can probably guess what this will do. What do you think will happen if you change the first line to "`x=101`", so that the condition tested by the `while` is untrue right at the start? Make a guess, and then try doing it. Was your guess right?

Challenge: Write a program that repeatedly asks you to enter a number, and keeps doing this until you enter the number 1234. You may need to know that to say "`x` is not equal to `y`" in Python, you write "`x <> y`".

Lots of guesses

So, now we know how to do something over and over using a `while` loop. How can we use this to make our program keep asking for guesses until the player guesses correctly?

You might at first think it should go something like this:

```
blah blah blah                          Choose a number
while guess <> number:
  blah blah                             Ask for a guess
  blah blah                             Say whether it was right or wrong
```

If you try writing your program this way and running it, you'll see that there's a problem. The variable `guess` (as I've called it above), that's supposed to be a name for the last number the player guessed, hasn't been set up when the machine first looks at the `while`. It only gets a value later on.

There are a couple of different ways to get around this. I think the simplest is as follows.

Make the loop say, not `while guess <> number:` or whatever, but simply `while 1:`. Remember that PYthon treats zero as meaning "false" and any non-zero number as meaning "true", so a loop that begins `while 1:` will go on for ever – it will never stop.

That doesn't seem like much of an improvement. But, try the following.

```
while 1:
  x = random_between(1,10)
  print x
  if x == 5:
    break
  print 'x was not 5!'
```

The only new thing here is the second-last line: `break`. If you try running that program you should be able to guess what `break` means. In case you can't, I'll tell you: it means "stop going round the loop you're in the middle of, right now". (What happens if a `break` happens while the computer is in the middle of *more than one* loop? Make a guess, and write a program to see whether your guess is right.)

Now that you know about `while` and `break`, you should be able to make your guessing-game program keep asking for guesses until the player enters the correct number. Just put the guess-asking bit inside a `while 1:` loop, and do a `break` if the player enters the right number.

How many guesses?

Keeping track of the number of guesses the player has made is easy. It's just like keeping track of the number of correct answers given in the tables-testing program in Sheet 2. Have a variable for it, set it to 0 at the start, and add 1 to it each time the player makes a guess.

Go on, do it.

Another game?

Now, this game is so exciting that when you've played it once you'll be desperate to play again (maybe). So when the game is over the machine ought to ask you if you'd like another.

You already know about the function `read_number()`, which lets the user enter a number. But what you want here isn't a number, but a yes-or-no answer to a question. To get that, use the function `read_yesorno()`. Try this:

```
>>> print read_yesorno('Would you like another game?')
Would you like another game? yes
1
>>> print read_yesorno('Would you like another game?')
Would you like another game? no
0
```

If it's not obvious yet what `read_yesorno` does, play with it some more: try giving it different questions to ask, and

answering differently. What happens if you just say `y` instead of `yes`? Does it matter whether you say `no` or `NO`? I've said this before, but it's worth saying again: *experiment*.

Of course, as well as printing the result of `read_yesorno` you could give it a name

```
bored = read_yesorno('Are you bored yet?')
```

or use it in something more interesting:

```
if read_yesorno('Would you like some chocolate?'):
  print 'Sorry, I have none.'
else:
  print 'Very strange.'
```

What we want to do is to keep on playing games, until the player says that they don't want another. We can do this by putting the whole program – everything you've written so far – inside another `while 1:`, and doing a `break` if the player says no to "Would you like another game?".

So, do that. Then check that your program still works.

What next?

There are plenty of ways to make this program better. Here are a few suggestions.

- Let the player decide what range the numbers should be in. There's nothing magical about having numbers from 1 to 1000; you might want a quicker game (1 to 20?) or a slower one (1 to 1000000000?).
- Make it a two-player game, so that one player enters a number and the other has to guess it. Clear the screen after the first player enters their number.
- If you do that, you could then make the players swap places each game, so that the player who chooses the number in one game has to guess the other player's number in the next game.
- And if you do *that*, you'll probably want to make the program begin by asking for the players' names, so that it can say at the start of the game whose turn it is to do what.
- The program could also keep track of how many guesses each player has needed on average.
- (This is harder!) Make the computer do the guessing – you choose a number, and answer the computer's guesses. To make this interesting, you'll need to think of a good strategy for the computer to use when guessing…

When you've done that, you could go on to Sheet 5, in which you'll write a more interesting game using the graphics stuff you learned about in Sheet 3.