2: Turning the Tables

Gareth McCaughan Revision 1.8, May 14, 2001

Credits

Introduction

One thing computers are very, very good at is arithmetic. This sheet will show you how to make the computer test how good you are at it. (The computer is probably about 2,000,000,000 times faster than you, but for now we'll only test how accurate you are.)

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 Sheet 1
- Some simple arithmetic!

Planning it out

When you start to think about writing a program, it's helpful to begin by thinking exactly what will usually happen when the program runs. It'll probably go something like this:

```
What's 6 times 7? 49
No, I'm afraid the answer is 42.
What's 3 times 2? 6
That's right -- well done.

What's 5 times 9? 45
That's right -- well done.

I asked you 10 questions. You got 7 of them right.
Well done!
```

And so on, with several more questions. . .

So, here are some things we need to be able to make the computer do:

- Choose numbers (at random, preferably)

- Display a sum
- Calculate the right answer
- Get an answer from the person using the program
- See whether it’s right or not
- Display a “that’s right” or “that’s wrong” message
- Keep count of how many questions were answered right
- Ask a total of (say) 10 questions, and then stop
- Display a final message saying how you’ve done

That’s quite a lot of things to do, but most of them are quite easy. We’ll do them one at a time.

Random numbers

Let’s start with the first item on that list: choosing the numbers. Try asking Python this and see what it says:

```
>>> from livewires import *                This is magic; see the box below.
>>> random_between(10,15)
```

What do you think `random_between` does? Try repeating that last line a few more times until you’re sure you understand how it behaves.

That first line there is important. You’ll need to put it at the start of every program you write. It lets you use some things we’ve added to the basic Python language, to make your life easier.
As well as putting it in programs you write, you should also type it in when you start a Python session. Otherwise Python won’t understand some of the things you ask it to do.
*See Sheet M (*Modules*) if you want the gruesome details of what’s going on here.*

Challenge: Make Python print a sum like

```
What is 7 times 2 ?
```

where the two numbers are generated randomly, between 1 and 10 (inclusive).

Questions and answers

OK, so now we know how to make Python display the question. This isn’t much use if we can’t get an answer from the ~~victim~~ person using the program. Fortunately, we can. Try this and see what happens:

```
>>> print read_number()
```

What about this?

```
>>> print 2*read_number()
```

Challenge: Make Python ask for *two* numbers, add them up, and print the result. Hint: you can do it with a single command.

You can change what Python says when asking for a number, if you like. Try this:

```
>>> print read_number('What is your favourite number?')
```

Now: Write a short program that displays a sum using two random numbers (as above), and then gets the user to type in an answer. This should just be a matter of combining two things you've already done. Don't worry about making the program check whether the user's number is right, or anything. Save this program so that you don't lose it: you'll be needing it again.

Remembering the numbers

Well, now is the time to worry about making the program check whether the user's number is right. This is trickier than anything else we've done so far, because we need to use those random numbers *twice*: once when we're saying what sum we want done, and once when we're working out the right answer.

Important Principle: If you're writing a program and you need to use something twice, *give it a name*.

We discussed names in sheet 1, but here's a brief reminder.

You give things names by using the = sign, and after that you can use the name instead of the thing you named:

```
>>> thing=1234
>>> 5*thing
6170
```

Challenge: Write a program that makes a random number and prints it out 5 times. (This is *not* the same thing as printing out 5 different random numbers!) You'll need to give the random number a name.

Challenge: Find the program you wrote that prints a sum and asks for the answer. Change it so that it gives names to the numbers in the sum. Check that you've done it right by making it print the sum twice – run the program and make sure that it does ask the *same* question twice, not two different questions.

Make your program do the following:

1. Choose two random numbers from 1 to 10.
2. Display a multiplication sum involving them.
3. Ask what the answer is.
4. Say what the answer should have been.

You've already done the first three of these; so the only new thing is the last one. Save the program again!

Right or Wrong?

The program you've just written could be used for testing someone's tables, but it seems rather silly to make the human check all the answers; that's just the kind of boring job computers do well. So, the next stage is to get the machine to check whether the answer you gave was right, instead of just saying what it should have been and leaving you to decide.

To do this, we need a new – and very important – idea.

If . . .

Write a program that looks like this, and run it. (The spaces at the start of some of the lines are important!)

Even though this isn't underlined, you should still type it in. The underlining thing is only done when there's a mixture of things you type in and things that the computer says, to help you see which is which.

```
if 1 < 2:
  print 'Something is wrong!'
else:
  print 'Something is right!'
```

The reason why the spaces at the start of a line matter is that Python uses them to decide how much of your program is controlled by the `if`*. Suppose you say*

```
if 1 > 1000:
  print 'Ouch!'
print 'Boink!'
```

Then the computer will *print* `Boink!`*, because that line isn't part of the* `if`*. But if the* `print 'Boink!'` *line had spaces at the start like the* `print 'Ouch!'` *line, then it* would *be part of the* `if`*, and therefore wouldn't get printed.*
Most computer languages don't take any notice of space at the start of a line. This means that they have to solve the problem in a different way; usually they need something like `end if` *at the end of the* `if`*, or else they insist that you surround everything controlled by the* `if` *in some kind of brackets. Python's way is easier to read.*

Now change the < to >, and run the program again. Can you guess what's going on? (In case you haven't already met the symbols in school: < means "is less than", and > means "is greater than".)

Challenge: Write a program that asks you for a number and then prints different things depending on whether the number is bigger than 100 or not.

For our tables-testing program, of course, what we want to know isn't whether the number you typed in is *bigger than* the correct answer; we want to know whether it's *equal to* it.

You might expect to do that by writing `if` *something* `=` *somethingelse*`:` and so on, but in fact it turns out that to avoid confusion between (1) using = to mean "is equal to" and (2) using = to mean "is the name of", Python uses different symbols for those two things. We've already seen that = is how you say "is the name of", so "is equal to" must be something different.

In fact, in Python "is equal to" is written with *two* equals signs, like this: `==` .

(If you want to know more about `if` and related things, see Sheet C (*Conditionals*).)

Challenge: Change the little program you just wrote that tests whether a number is bigger than 100, so that instead it tests whether the number is *equal to* 100.

Now you should have no difficulty making your tables-testing program print a "yes!" or "no!" message depending on whether you typed in the right answer to the sum or not. So now you have a program that tests you on *one* multiplication sum, and then stops. We're making progress…

Over and over and over again

You may remember the `for` loop, from sheet 1. Whether you do or not, here's an example of how to use it. Write a little program:

```
for x in 1,2,3,4,5,6:
  print 'x is', x
```

(Do I still need to remind you that *spaces at the starts of lines matter*?)

Challenge: Put that whole program inside another `for` loop, so that it all happens three times.

If you managed that, find your tables-testing program and do something similar so that it does everything 10 times.

If you want to know more about how to do things over and over again in Python, see Sheet L (*Loops*).

Who’s counting?

A little while ago I was complaining that our program forced the person using it to check their own answers. We’ve fixed that now, but the user still has to count how many answers they’ve got right. The computer ought to be able to do that.

Here’s a little program to try. It doesn’t have much to do with the tables-tester, but you’ll probably find what you have to do next easier if you try this first.

```
odd=1
for x in 1,2,3,4,5,6:
  print ’An odd number:’, odd
  odd = odd+2
```

That last line may look rather strange: how can `odd` be equal to `odd+2`? Well, remember that I said a little while ago that in Python `==` means “is equal to”, and `=` means “is a name for”. What the line tells Python to do is: work out `odd+2`, and then call *that* `odd`.

(In sheet 1 I mentioned that what we’re calling “names” are often called “variables”. You’ve now discovered why: the things they name can change, or *vary*. In the program above, `odd` starts off meaning 1; then it means 3, then 5, and so on.)

OK. Back to the tables-testing.

Add the following lines to your tables-testing program. You’ll have to work out *where* in the program. Some of the lines may need some spaces added at the start.

```
right = 0
wrong = 0
right = right + 1
wrong = wrong + 1
print ’You got’, right, ’questions right and’, wrong, ’wrong.’
```

Improving the program

If you’ve done everything correctly so far, you now have a program that does roughly what I described at the start of this sheet. There are lots of things that could be made better; if you aren’t fed up of the program yet, you could try some of these:

1. *Improve the layout*. It’s a pity that the question and the user’s answer have to be on different lines, and that there are some unnecessary spaces. Have a look at Sheet I which tells you a bit more about “input and output”: that is, making the program ask things or say things. You may find that you also need to know some things described in Sheet S (*Strings*).
2. *Timing*. It’s probably not hard to get 100% if you don’t mind taking, say, an hour over each question. But if you get 100% and take less than a second per question, you’re doing very well. So, make the program time you and report at the end how long you took. You’ll need to look at Sheet T (*Time*) for this.
3. *Adjustable difficulty*. Some people are very good with numbers. They might find being tested on numbers from 1 to 10 rather boring. Some people are very bad with numbers, and might prefer easier questions. Obviously, it’s not too hard to change the program to use larger or smaller numbers. (Look at the program and make sure you can see how to do that.) It might be more interesting if the program became a little harder every time you get a question right, and a little easier every time you get one wrong. (For this to work well, you’d probably need to ask more than 10 questions.)
4. *Adjustable length*. You might want a quick test, with only four questions. Or a long one, with 100 questions, to see how long you can stay awake. Make the program begin by asking how many questions you want, and then ask that many.

   To do this, you’ll need to know about things called “ranges”. They’re described in Sheet L (*Loops*).

What next?

The next sheet in the series is sheet 3, “Pretty pictures”. It’s all about graphics: drawing pictures with Python.