LINUX VOICE TUTORIAL

CODE NINJA: PROGRAMMER'S GOLF

BEN EVERARD

Sometimes you just have to prove, without a doubt, that you're the best programmer in the room.

WHY DO THIS?

- Show off your 1337 programming skillz.
- Push yourself to learn more about your language of choice.
- Save minuscule amounts of disk space.

People are naturally competitive. There's just something in human nature that makes us want to find out who's the best at something, whether it's who's the fastest runner, who can jump the furthest or who's the best at kicking balls between goalposts. Sometimes we geeks like to think we've transcended this base desire. Perhaps you have, but many of us have just transferred this competitive instinct from physical exploits to mental ones.

Linguists have crosswords, mathematicians have number puzzles, and techies have programmer's golf. Programmer's golf in case you're wondering, is the challenge of taking a particular problem and coding it in as small a number of characters as possible.

There aren't any fixed rules for this other than the result must be accepted by the interpreter or compiler as a valid program, and sometimes there are restrictions on the modules or libraries that can be used. Beyond that, anything is permissible.

A good understanding of the language being used is essential, especially as it's often the language's more esoteric features that can result in saved space.

Let's take a look at a simple example, printing the numbers 1 to 6 at one per line in Python. Done normally, this might look something like this:

```
for number in range(1,7):
        print number
```

This is 40 characters. It's easy to see we've wasted quite a bit on the variable name, so we can shrink this down to 30 characters by simply replacing it with a single letter:

```
for i in range(1,7):
        print i
```

If you're familiar with Python, you might know that there are two extra characters that we can get rid of quite easily.

```
for i in range(1,7):print i
```

It's clear that we need the **print** statement, because there's no shorter way of outputting text onto the screen. Of the remaining code, the **range** call takes up 8 characters, so it seems like a good place to look for further shrinking. We need something that Python can iterate over that returns the 1 to 6. It's important to realise that in this case, it doesn't matter if it's the numbers 1 to 6 or the characters 1 to 6, because **Python**'s print statement can work with either.

How short is a piece of string?

Once you've realised that it can be the characters 1 to 6, it's fairly obvious that we can iterate the **for** loop over a string instead:

```
for i in '123456':print i
```

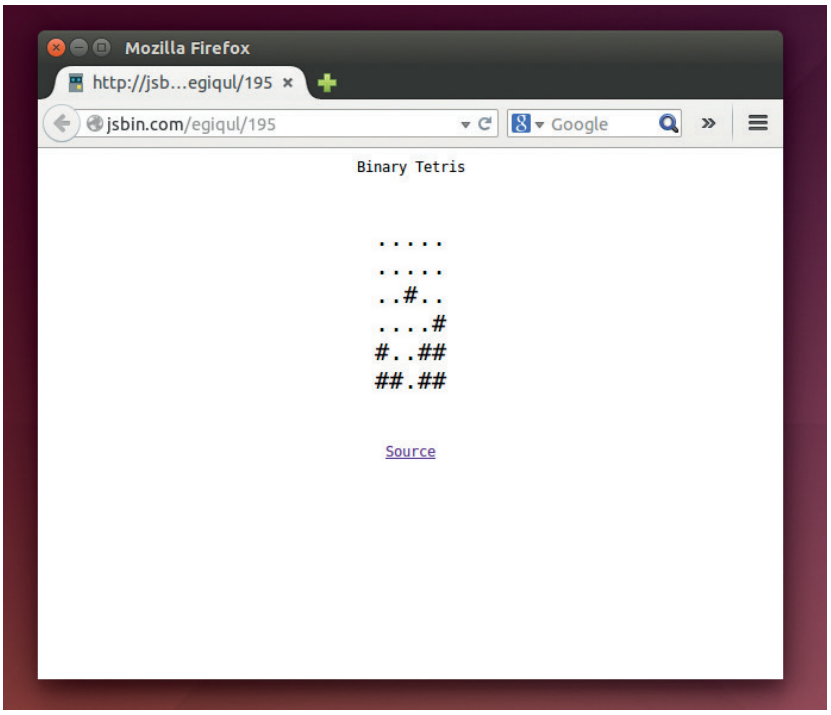
This has managed to claw back another couple of characters. What's more, it now uses the string type, which has quite a few powerful methods that perhaps we can make use of.

We're confident that **print** is the shortest way of outputting something to the screen, and we think that the string is the shortest way of encoding the numbers we need. The only place left to look is the **for** loop. Here, we need to think back to what the original challenge was: print the characters 1 to 6 with each character on a separate line. So far, we've been using a separate **print** statement for each line, and this has required us to use a loop to call the **print** statements on each number in turn. However, we could get rid of the loop if we printed them all with the same **print** statement, but put a new line character in-between each number.

```
print'\n'.join('123456')
```

This uses Python's **join** method on the string **'\n'**. This iterates through the argument and outputs every item in the argument with the original string between it. Since strings are iterated through on individual characters, this outputs:

The source code to the main engine of this simplified game of Tetris is: **function(a,b,c,d,e){return d+=c,e=a|b<<d,d<0|a&b<< d&&(a=e=parseInt((a|b<<c).toString(d=32).replace(/v/,""),d),b=new Date%2?1:3),[a,b,d,e]}**



```
Binary Tetris

.....
.....
..#..
....#
#..##
##.##

Source
```

```
1\n2\n3\n4\n5\n6
```

Since **\n** is the new line character, printing this results in each number being printed on a separate line.

There is another way of getting the code this short. In Python, you don't need to separate bits of text with spaces if the interpreter can distinguish between them, so you can remove the space between **in** and the start of the string. In other words, with:

```
for i in'123456':print i
```

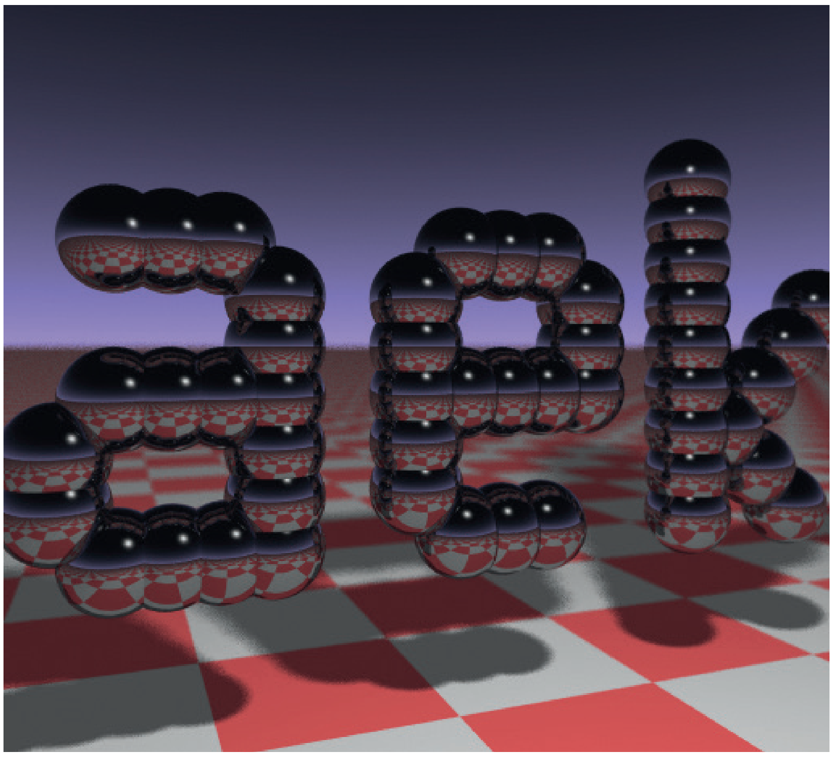
Is this as short as it can go? Possibly not. There's no way to know for sure that there definitely isn't a shorter way of doing something. If you find a way to remove a few characters, drop me an email at **ben@linuxvoice.com**. I'd love to hear it.

Some people may wonder why bother with this at all. After all, the resulting code is almost always an unreadable, unmaintainable mess. Wouldn't it be better to focus our competitive instincts on more useful aspects of programming like readability or performance? No decent programmer would focus their efforts on squeezing every last byte out of their code without a very good reason.

However, aside from the competitive aspect of the challenge, there are some skills to be learned in shrinking file sizes. For one, it forces you to learn more about your chosen language. For example, it's perfectly possible to program in Python for years, yet never really get to grips with lambda functions. However, if you're looking to squeeze a few characters out, they can be a fertile source of reductions. The features you learn may well help you program better in ways other than file size.

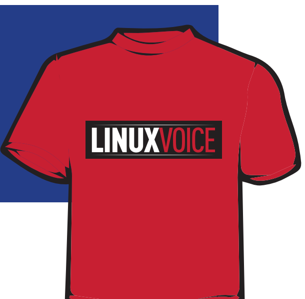
Many times, the tricks that you use to remove unnecessary bloat are quirks and edge cases of the language, and these can sometimes lead to bugs or other unexpected behaviour. Learning to exploit these means learning to understand them, and this means a better understanding of the language.

Now, let's see how good you are with a little competition. Fore!


The code used to create this 3D render is small enough to fit on a business card, and perhaps more impressively, is 1337 bytes long. For more details see **www.fabiensanglard.net/rayTracing_back_of_business_card**

COMPETITION

Write ridiculously small code, win an attractive garment!



This month, the Linux Voice challenge is a game of Python programmer's golf. Your challenge is to write the smallest possible Python program that takes a number as input, and prints the value in Roman numerals. To get you started, here's a sample program that does just this:

```
symbols = [('M', 1000), ('C M', 900), ('D', 500),
        ('C D', 400), ('C', 100), ('X C', 90), ('L', 50),
        ('X L', 40), ('X', 10), ('I X', 9), ('V', 5),
        ('I V', 4), ('I', 1)]

def romannumeral(number):
        while number > 0:
                for symbol, value in symbols:
                        if number - value >= 0:
                                print symbol,
                                number = number - value
                                continue
number_in = raw_input("Enter a number: ")
romannumeral(int(number_in))
```

This is obviously not optimised for size, so you shouldn't have too much trouble stripping some fat off it. The question is, how much?

There are a couple of things to point out about this code. It puts a space in between each character. This is for simplicity, and any spacing between characters other than new lines is acceptable as long as it's consistent.

There is also some contention about what the Roman numerals for certain numbers are. For example, should 1999 be MIM or MCMXCIX? Without wanting to get into a historical argument about how people would have written numbers thousands of years ago, we'll simply say that your program should match the form of Roman numerals given by our program.

Beyond this, there are just a few rules:

- The length of the code will be the total length of the submitted code in characters, and the person who submits the shortest code will win an exclusive Linux Voice winner's T-shirt.
- No modules can be imported. That would just make it too easy.
- Either Python 2 or 3 is acceptable.
- Email your entries to ben@linuxvoice.com by the end of the day on 15th October 2014.
- In the event that more than one person has an entry the same length, they will both be considered winners, but the first entry received will win the T-shirt.
- All code must be released under an OSI approved open source licence, and GPL v3 is preferred.