Exercise 1.

Use the bisection function as it is provided in `finding_zeros.py` and explain why the following generates an infinite loop

```
bisection(cos, 1e6*pi, (1.e6+1)*pi)
```

Try to change the default tolerance. What happens? Change the exit criterion in the code in such a way as to fix this problem.

Exercise 2.

Find a function that has a zero that can be found with Newton's method, but not with the bisection method. Hint: remember that a zero is bracketed in (a, b) if $f(a)f(b) < 0$. Can any zero be bracketed?

Exercise 3.

The function $f(x) = xe^{-x}$ has only the obvious zero for $x = 0$. If you use Newton's method with $0 < x_0 < 1$ it'll work like a charm. Likewise for negative starting points, provided that they are not so big (in modulus) as to overflow the exponential function (e.g. `exp(-(-1000))=inf`). The method obviously won't work for $x_0 = 1$, because that's a zero of the derivative. Explain why it doesn't work for, say, $x_0 = 2$.

Can you fix the problem and make the method work?

If you answer yes, write a version of the code that finds the zero. If you answer no, explain why. In this case would you keep the code as it is, or would you still do some modification? What exactly?

Exercise 4.

A method called "*Regula falsi*" attempts to get the best of both worlds: guaranteed convergence of bisection and faster–than–linear convergence of Newton's method. Look it up in the book, sec.1.5.1. Write and test a function that implements it. Then count how many iterations you need to find the zero of $f(x) = xe^{-x}$ with an accuracy of ten decimal digits, using bisection, Newton and regula falsi. Use the initial bracket $a = -1$, $b = 0.5$, or, for Newton, the initial point $x_0 = -1$.

Exercise 5.

In the Newton's method code modify the exit criterion from:

```
if fabs(a-newa) < tol:  break
```

to:

```
if fabs(a-newa)/fabs(newa) < tol:  break
```

This is a criterion based on the relative error, rather than on the absolute one, and it is in general a good idea (see exercise 1). However, explain why you get an infinite loop for

```
Newton(sin, cos, 1.)
```

but not for

```
Newton(sin, cos, 3.)
```

(in this case you find very quickly a good approximation of π).