Functions

Introduction to Functions and their syntax in Python.

We spent a lot of time earlier in part 1 of the guide working with mathematical functions. We thought of these as machines which take input, do some work, and pop out the result. And those functions stood in their own right and could be used again and again.

Many computer languages, Python included, make it easy to create reusable computer instructions. Like mathematical functions, these reusable snippets of code stand on their own if you define them sufficiently well, and allow you to write shorter more elegant code. Why shorter code? Because invoking a function by its name many times is better than writing out all the function code many times.

And what do we mean by “sufficiently well defined”? It means being clear about what kinds of input a function expects, and what kind of output it produces. Some functions will only take numbers as input, so you can’t supply them with a word made up of letters.

Again, the best way to understand this simple idea of a *function* is to see a simple one and play with it. Run the following code.

```
# function that takes 2 numbers as input
# and outputs their average

def avg(x,y):
    print('first input is', x)
    print('second input is', y)
    a = (x + y) / 2.0
    print('average is', a)
    return a
```

Let's talk about what we've done here. The first two lines starting with # are ignored by Python but for us can be used as comments for future readers. The next bit `def avg(x,y)` tells Python we are about to define a new reusable function. That's the "def" keyword. The "avg" bit is the name we've given it. It could have been called "Banana" or "Pluto," but it makes sense to use names that remind us what the function actually does. The bits in brackets *(x,y)* tells Python that this function takes two inputs, to be called *x* and *y* inside the forthcoming definition of the function. Some computer languages make you say what kind of objects these are, but Python doesn't do this, it just complains politely later when you try to abuse a variable, like trying to use a word as if it was a number or other such insanity.

Now that we've signaled to Python that we're about to define a function, we need to actually tell it what the function is to do. This definition of the function is indented, as shown in the code above. Some languages use lots of brackets to make it clear which instructions belong to which parts of a program, but the Python designers felt that lots of brackets weren't easy on the eye and that indentation made understanding the structure of a program instantly visual and easier. Opinions are divided because people get caught out by such indentation, but I love it! It's one of the best human-friendly ideas to come out of the sometimes geeky world of computer programming!

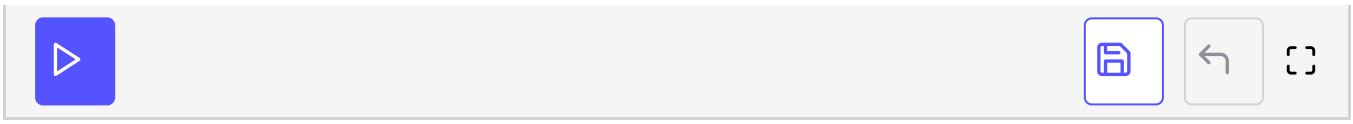
The definition of the `avg(x,y)` function is easy to understand as it uses only things we've seen already. It prints out the first and second numbers which the function gets when it is invoked. Printing these out isn't necessary to work out the average at all, but we've done it to make it really clear what is happening inside the function. The next bit calculates $(x + y)/2.0$ and assigns the value to the variable named a. We again print the average just to help us see what's going on in the code. The last statement says `return a`. This is the end of the function, and it tells Python what to throw out as the function's output, just like machines we considered earlier.

In the next line, write `avg(2,4)` to invoke this function with the inputs 2 and 4. By the way, invoking a function is called *calling a function* in the world of computer programming. The output should be what we expect, with the function printing a statement about the two input values and the average that it calculated. Look at the following cell and run it after adding your code below the comment. You can test this code with different inputs.

```
def avg(x,y):
  print ("frist input is", x)
  print ("second input is", y)

  a = (x+y) / 2.0
  print ("average is", a)
  return a;

#write your code here
```



You may have noticed that the function code which calculates the average divides the sum of the two inputs by 2.0 and not just 2. Why is this? Well, this is a peculiarity of Python which I don't like. If we used just 2 the result would be rounded down to the nearest whole number, because Python considers just 2 as an integer. This would be fine for `avg(2,4)` because $6/2$ is 3, a whole number. But for `avg(200,301)` the average is $501/2$ which should be 250.5 but would be rounded down to 250. This is all just very silly I think, but worth thinking about if your own code isn't behaving quite right. Dividing by 2.0 tells Python we really want to stick with numbers that can have fractional parts, and we don't want it to round down to whole numbers.

Let's take a step back and congratulate ourselves. We've defined a reusable function, one of the most important and powerful elements of both mathematics and in computer programming.

We will use reusable functions when we code our own neural network. For example, it makes sense to make a reusable function that does the sigmoid activation function calculation so we can call on it many times.