Object-oriented programming - Day 2

March 23, 2015

1 Simple class - String

1. Write a class `String` which consists of only one method, the constructor. Apart from the obligatory `self`, the constructor shall take one additional argument `s`. In the constructor initialize a member variable of the class, let's call it `mystring`, with the value of `s`.
2. Add an additional method to the class. This method shall also take one argument called `other` (+ `self`). In the function body check if `other` is contained in `mystring`. If so return `True`, otherwise return `False`. *Hint*: Try the `in` keyword.
3. Add a class (`static`) variable called `counter` to the class and initialize it with 0.
4. Change the initializer such that it increments `counter` by one.
5. Create several instances of this `String` class and check afterwards how many you have.
6. Add a 'destructor' function `__del__` which decreases `counter` by one. Test it, i. e. inspect the value of counter.

2 Inheritance - Animal

1. Write an `Animal` base class. Each animal shall have an `age` and a `weight`. Once set in the constructor these variables should not be changeable from the outside (you cannot just change the age of an animal as you like, right?). But provide functions that allow to read the values of these variables. *Note:* In Python "private" attributes are not really private. But it is the best protection of member attributes against the outside that we can get — so we take it.

2. Implement a `speak` and a `move` method which print an error message (an abstract animal can neither speak nor move).

3. Write a `Cat` and a `Fish` class which inherit from `Animal`. Override the `speak` and `move` methods such that they print out an appropriate message when called (something like "Meow" for the cat. . .)

4. Test your implementation in the python interpreter by creating instances of `Cat` and `Fish`.

5. Try to make sense of what's happening here:

```
> c = Cat(2, 3.7)
> c.speak( )
Meow.
> Animal.speak(c)
I am an abstract animal and cannot speak.
> num = 3.7
> Animal.speak(num)
[some peculiar Python−error message]
```

6. Add an `eat` method which takes as an argument the object to eat; for exam- ple: `mycat.eat(myfish)`. Obviously the fish should be dead afterwards, so the method's purpose is to "kill" the fish, i.e. the fish should not be accessible any more (the `myfish` instance should be deleted). How could you accomplish this?

3 An integrate-and-fire neuron

A more complex example: using what we've learned so far

The equation determining the membrane potential of a *leaky integrate and fire neuron* is given by

$$c_M \dot{V} = -g_L \left(V - V_L\right) + i_{ext}$$

where c_M is the membrane potential, g_L the leak conductance, V_L the corresponding reversal potential and i_{ext} an external current that drives the neuron. In addition, whenever V becomes larger than a threshold value V_{th} a spike is elicited and V is reset to a value V_r.

1. Write a class which is initialized with the necessary parameters (like $c_M, \ldots,$ don't forget an initial value for V). Except for the external current it should not be possible to change the neuron parameters after instanciation. Cf. the hint in exercise 2.1.

2. Using the Euler method

   $$\dot{V} \approx \frac{V(t+h) - V(t)}{h}$$

   we can derive the following update rule:

   $$\begin{aligned} V(t+h) &= V(t) + h\dot{V}(t) && (1) \\ &= V(t) + \frac{1}{c_m}\left(-g_L \left(V - V_L\right) + i_{ext}\right) && (2) \end{aligned}$$

   Implement a method which updates V according to this rule until a time T has passed. In each step append the newly calculated V to a list `trace`.

3. Now consider that V is reset everytime it crosses the threshold.

4. Add a method that uses *matplotlib* to plot the voltage trace. As *matplotlib* will be treated later, here's a spoiler: import matplotlib.pyplot as plt plt.plot(xvals,yvals)

5. Test your neuron for different initial values for V and different external currents i_{ext}.

6. *If you are quick:* Simulate a network of neurons. Each neuron is connected to some others (e.g. with a predefined probability). The equation to simulate is now

   $$c_M \dot{V_j} = -g_L \left(V_j - V_L\right) + i_{ext} + i_{j,syn}$$

   where

   $$i_{j,syn} = \sum_k \sum_m w_{jk} K(t - t_{jk}^{(m)})$$

   and $K(t) = \delta_{t,0}$. $w_{j,m}$ are weight factors specifying the strength of the connection between neuron m and j and $t_{jk}^{(m)}$ is the time when neuron j receives its m-th spike from neuron k. In other words: whenever neuron j receives a spike from neuron k its voltage changes by an amount w_{jk}.

4 The Zoo

Dictionaries, special methods, random numbers, functions as arguments to functions

We want to collect some animals (the ones from the `Animal` exercise above) in a `Zoo` object. It will consist of a number of animals and each of them should have a name. 1. Provide a way to add new animals to the `Zoo`. 2. *Special methods.* Implement some special methods like - the length-operator `len(self)`, to see how many inhabitants the zoo has. Test with `len(myzoo)` (where `myzoo` is an instance of `Zoo`) - the "less than" operator `lt(self,rhs)` which should return `True` if `self<rhs` and `False` if `self>=rhs`. Test with `myzoo1 < myzoo2` ... - the subscripting operator `getitem(self, name)` to access an animal from the zoo by name (e.g. `z["blub"].speak()`), - … 3. The zoo welcomes 1000 new animals. As the administration can't come up with so many names at the same time, they shall be named, for the moment, by numbers. Write a method which adds a number `n` of new animals of different species, age and weight. You can use the functions `random, randint` and `randrange` from module `random` to create random numbers. 4. Make it possible to rename animals. 5. For easy book-keeping add a `select` method which takes one argument `fltr`. `fltr` is itself a function accepting an `Animal` instance as argument and returning a boolean. `select` returns a list of all animals for which the `fltr` function returns `True`. Test this function: - select all animals which are younger than 2 - select all animals for which `age+weight` $\leq \pi$ - select all `Cats` 6. Let the zoo visitor specify a simple(!!!) criterium for animal selection. I. e. create a method `visitorSelect` with a string input argument which evaluates it and then calls `select`.

5 Temperature

Write a class that stores a temperature in one unit and allows accessing it in several other ones (cf. http://en.wikipedia.org/wiki/Conversion_of_units_of_temperature for conversion formulas).

Hint: Have a look at `__getattr__` , `__setattr__` to access and set temperatures.

6 Vector

A `list` or `tuple` can be used to store floating point numbers. They are, however, not suitable as vector classes. Namely, apart from performance issues, it would be desirable if one could do basic calculations, such as addition by writing `z = x + y`, where `x, y, z` are instances of such a vector class. (Note that a `list` has an operator + but it does something else!). Therefore write a `Vector` class which allows basic vector space operations (e. g.: addition, subtraction, scalar multiplication and division, dot product, …).