Dynamic Languages - Übungen Blatt 4

Abgabedatum: `17. Mai 2010`

Aufgabe 1 - Proxies

(3 Punkte)

Implement a logging proxy class. Its constructor takes an arbitrary object. The proxy object is supposed to behave as much as possible as the original object. In addition, the proxy is supposed to keep a list of all attribute that were accessed on the object, including special methods, in order of access. The log can be accessed using a global function `get_proxy_log`. Some test functions are given in `blatt4.py`.

In addition, write another test function with a proxy around a dictionary, testing at least the special methods `__getitem__` and `__setitem__`.

Ideally, your source code should not define all `__xxx__` methods by copy-pasting them in the source of the class, but by putting these methods inside the class programmatically (with a loop).

Aufgabe 2 - Prototypes

(4 Punkte)

The point of this exercise is to explore a different approach to object-oriented programming, called prototype-based programming. In prototype-based programming there is no distinction at all between classes and objects. The task of this exercise is to implemented a class `ProtoObject` in Python whose instances behave like prototype-objects. This behaviour includes:

- Every prototype-object has a number of attributes. Among those attributes there is a special attribute called `parent` which must be another prototype-object or `None`.
- When trying to read an attribute from a prototype-object, first the attributes of the object itself are checked, then the attributes of the `parent` object, and so on.
- When reading an attribute, and that attribute's value has a `__get__` method, that method should be called to get proper binding behaviour.
- Writing an attribute works different from Python: When writing an attribute the `parent` chain is searched for a place where the attribute already exists. The new value is put into this prototype-object. When the attribute exists nowhere yet, it is put into the original object.

- If the `parent` attribute is not specified, it is set to a base prototype-object called `default_parent`.
- The `default_parent` object has just one attribute, a method `clone` that can be used to make a copy of a prototype-object.

In addition to `ProtoObject` implement a metaclass `ProtoMeta` that can be used to build prototype-objects using the class-syntax. ProtoMeta needs to implement `__call__` and `__call__` should return a new instance of `ProtoObject`.

The file `blatt4.py` contains a number of tests for prototype-objects. If you hit under-specified behaviour during the implementation, decide on a sensible behaviour and write a test for it.

Aufgabe 3 - Lua tables

(3 Punkte)

The Lua programming language has a single data structure called *table* instead of Python's *list* and *dict*. A table is a dictionary-like structure that can also behave like a list when the keys are integers.

The purpose of this exercice is to design, test and implement a simple `Table` class in Python. It should have a dictionary-like interface, supporting at least expressions like:

- `table[key] = value`
- `value = table[key]`
- `key in table` (test for existence of a key; this calls the `__contains__` special method on the table)

It should also support list-like expressions like:

- `len(table)`, which should return the smallest integer `n` such that `n not in table`. For example, if the table contains the keys 0, 1 and 2, then its length should be 3. If the table also contains other keys like 42 and "hello world", then they are ignored the length is 3 anyway. If the table doesn't even contain the key 0, its length is 0.
- `table.append(x)`, equivalent to `table[len(table)] = x`.
- `del table[index]` and `table.insert(index, x)` should work like in lists: they should remove or insert an element in the middle of the list, shifting the end of the list. For example, if the table contains `{0: 'x', 1: 'y'}`, then after `del table[0]` it should contain `{0: 'y'}`.
- `table1 + table2` should work like list concatenation if the tables just contain consecutive 0-based numbers as keys. For the cases of tables that also contain more keys, design a "sensible" way to put them in the result.

Write tests *first!*