U6944D5GU : :eyes:
U5NMSURAQ : the function applies to every element of the original array
U5NMSURAQ : so you see three plots: `y = x^1`, `y = x^2` and `y = x^3`
U5ZPMJA06 : I have here a very small unit test program which is supposed to test that an object throws an AttributeError exception when a nonexistent
attribute lookup is attempted.

```
"""
It works fine, but I had to put the attribute access in a function.
I'd rather not define that function and use the assertRaises as follows:

  self.assertRaises(AttributeError, lambda item: item.donation)

However, this gives me a "TypeError: &lt;lambda&gt;() takes exactly 1 argument (0 given)"

Is this somehow possible without helper function?
"""

import unittest

class Person(object):
    pass


class MyTestCase(unittest.TestCase):

    def testNonexistentAttibute(self):
        def bombfunc():
            p = Person()
            p.name = "Joe"
            p.money = 2800
            p.money += p.donation # boom!
        self.assertRaises(AttributeError, bombfunc)
        self.assertRaises(AttributeError, lambda item: item.donation) # How to make this work?


if __name__  == "__main__":
    unittest.main()
```

U5LNXQHN3 : I know you can use a `with` block in some other test packages, but to be honest this seems like a weird thing to be testing for
U5LNXQHN3 : You could probably use `self.assertIn("donation", item.__dict__)` or some similar abomination if you really want a one-liner
U5ZPMJA06 : <@U5LNXQHN3> Yeah it sounds weird, but actually the object under test is some kind of container giving both attribute and keyed lookup access to a set of properties, and I need to test whether the right exceptions are thrown during lookup of a nonexistent property.
U5LNXQHN3 : Sounds like a bad idea to me. But if I had to write tests for it, I'd just use the __dict__ check directly. Or `hasattr`.
U5ZPMJA06 : <@U5LNXQHN3> Bad idea? You have a better idea? Thanks for the `hasattr` tip! This is what it's all about, the `Bunch` object: <https://github.com/motoom/bunch>
U5LNXQHN3 : I think that is unnecessarily blurring the lines between a container and a type. If you don't know what attributes a type has, then you don't really know what interface it provides, which makes it a very awkward object to work with
U5ZPMJA06 : <@U5LNXQHN3> I use it declutter my source code. Basically it is a `dict` like object where you don't have to type `["` and `"]` all the time. So I can write:```
U5ZPMJA06 : ```for r in bunched(recordset): # Where recordset is fetch_all() of DictCursor   if r.salary < 3000:
      print r.name, "could use a raise"
```
```

U5ZPMJA06 : Psycopg2 has a `NamedTupleCursor`, which provides the same syntax.
U5LNXQHN3 : In that case, at least the schema is documented elsewhere. But I don't like it.
U5ZPMJA06 : The alternative would be:```
for r in recordset:
  if r["salary"] < 3000:
      print r["name"], "could use a raise"
```
You prefer that?

U5LNXQHN3 : Maybe
U5ZPMJA06 : I can work like that. You have a job for me? :stuck_out_tongue_winking_eye:
U5LNXQHN3 : In the general case, yes. It's not exactly a great hardship. In a database context, it might be nice to have a type that directly reflects the DB schema - which is what we have ORM for
U5ZPMJA06 : btw, the `NamedTupleCursor` gets its attribute names from the fields behind the *SELECT*. I.e. `select name, salary from Employees` would result in *.name* and *.salary* attributes on the tuples in the resultset.
U5LNXQHN3 : I just really dislike things that attempt to cut down on a bit of typing by ruining the interface. Several parsers do it, like BeautifulSoup. lxml.objectify is even worse
U5ZPMJA06 : I wouldn't call it ruining. It makes the code more readable to me. I prefer dot notation over index lookup with a string.
```