- Extra
    - **Total Stats**
        - **217** Posts
        - **544** Tags
        - **556** Comments
        - **264** Comment Posters

May 3, 2009

"All methods in Python are effectively virtual"

Filed under: Blog — krkhan @ 8:07 pm

Dive Into Python really is one of the best programming books I have ever laid my hands on. Short, concise and to-the-point. The somewhat unorthodox approach of presenting an alien-looking program at the start of each chapter and then gradually building towards making it comprehensible is extraordinarily captivating. With that said, here's an excerpt from the chapter introducing Python's object orientation framework:

> Guido, the original author of Python, explains method overriding this way: "Derived classes may override methods of their base classes. Because methods have no special privileges when calling other methods of the same object, a method of a base class that calls another method defined in the same base class, may in fact end up calling a method of a derived class that overrides it. (*For C++ programmers: all methods in Python are effectively virtual.*)" If that doesn't make sense to you (it confuses the hell out of me), feel free to ignore it. I just thought I'd pass it along.

If you were able to comprehend the full meaning of that paragraph in a single go, you most definitely are one of the following:

- Guido van Rossum himself
- Donald Ervin Knuth
- Pinocchio

Neither of which happens to be my identity, so it took me around three rereads to grasp the idea. It brought back memories of an interesting question that I used to ask students while I was working as a teacher's assistant for the C++ course: "What is a virtual function?" The answer *always* involved pointers and polymorphism; completely ignoring any impact virtual functions would be having on inheritance in referential/non-pointer scenarios. (Considering that most of the C++ books never attempt to portray the difference either, I didn't blame the students much.) Confused again? Here's some more food for thought: Python does not even have pointers, so what do these perpetually virtual functions *really* entail in its universe? Let's make everything peachy with a nice example.

Consider a `Base` class in C++ which defines three functions:

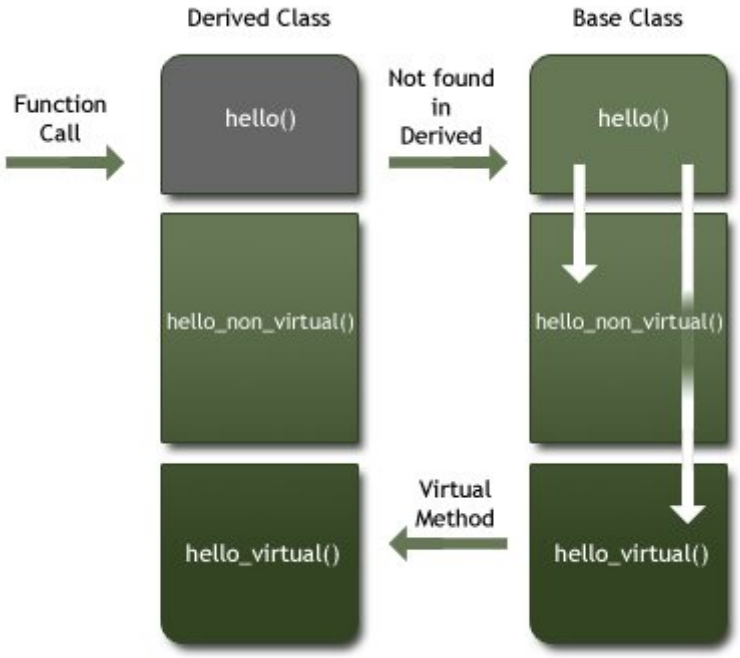
- `hello()`
- `hello_non_virtual()`
- `hello_virtual()`

The first function, i.e., `hello()` calls the latter two (`hello_non_virtual()` and `hello_virtual()`). Now, we inherit a `Derived` class from the `Base`, and override the functions:

- `hello_non_virtual()`

- `hello_virtual()`

Note that the `hello()` function is **not** defined in the `Derived` class. Now, what happens when someone calls `Derived::hello()`? The answer:



Since `Derived::hello()` does not exist, `Base::hello()` is called instead. Which, in turn, calls `hello_non_virtual()` and `hello_virtual()`. For the non-virtual function call, the `Base::hello_non_virtual()` function is executed. For the virtual function call, the overridden `Derived::hello_virtual()` is called instead.

Here's the test code for C++:

```
#include <iostream>

using namespace std;

class Base {
public:
        void hello()
        {
                cout<<"Hello called from Base"<<endl;

                hello_non_virtual();
                hello_virtual();
        }

        void hello_non_virtual()
        {
                cout<<"Hello called from non-virtual Base function"<<endl;
        }

        virtual void hello_virtual()
        {
                cout<<"Hello called from virtual Base function"<<endl;
        }
};

class Derived : public Base {
```

```
27 public:
28         void hello_non_virtual()
29         {
30                 cout<<"Hello called from non-virtual Derived function"<<endl;
31         }
32
33         void hello_virtual()
34         {
35                 cout<<"Hello called from virtual Derived function"<<endl;
36         }
37 };
38
39 int main()
40 {
41         Derived d;
42
43         d.hello();
44
45         return 0;
46 }
```

And its output:

> Hello called from Base
> Hello called from non-virtual Base function
> Hello called from virtual Derived function

Similarly, a Python program to illustrate the statement *"all methods in Python are effectively virtual"*:

```
 1 class Base:
 2         def hello(self):
 3                 print "Hello called from Base"
 4
 5                 self.hello_virtual()
 6
 7         def hello_virtual(self):
 8                 print "Hello called from virtual Base function"
 9
10 class Derived(Base):
11         def hello_virtual(self):
12                 print "Hello called from virtual Derived function"
13
14 d = Derived()
15 d.hello()
```

Output:

> Hello called from Base
> Hello called from virtual Derived function

I hope this clears up the *always-virtual* concept for other Python newcomers as well. As far as my experience with the language itself is concerned, Python is sex; simple as that. Mere two days after picking up my first Python book for reading, I have fallen in love with its elegance, simplicity and overall highly addictive nature.

Tags: C++, Class, Code, Dive Into Python, Example, Flag 42, Function, Method, Object Oriented Programming, OOP, Open Source, Polymorphism, Python, Virtual
Comments (3)