- Updates

- Inspirated
  - Blog
  - Code
- Pages
  - about
- Archives
  - November 2016 (1)
  - May 2016 (1)
  - June 2015 (1)
  - April 2015 (1)
  - July 2014 (1)
  - June 2014 (1)
  - July 2013 (1)
  - December 2012 (1)
  - November 2012 (1)
  - March 2012 (1)
  - December 2011 (1)
  - November 2011 (1)
  - October 2011 (3)
  - September 2011 (1)
  - August 2011 (2)
  - July 2011 (2)
  - June 2011 (3)
  - May 2011 (1)
  - April 2011 (3)
  - March 2011 (1)
  - February 2011 (1)
  - November 2010 (1)
  - October 2010 (2)
  - September 2010 (3)
  - August 2010 (2)
  - July 2010 (5)
  - June 2010 (9)
  - May 2010 (8)
  - April 2010 (3)
  - March 2010 (3)
  - February 2010 (7)
  - January 2010 (1)
  - December 2009 (1)
  - November 2009 (6)
  - October 2009 (4)
  - September 2009 (2)
  - August 2009 (7)
  - July 2009 (7)
  - June 2009 (3)
  - May 2009 (9)
  - April 2009 (3)
  - March 2009 (7)

- February 2009 (7)
- November 2008 (2)
- October 2008 (2)
- September 2008 (6)
- August 2008 (2)
- July 2008 (9)
- June 2008 (5)
- May 2008 (1)
- April 2008 (1)
- March 2008 (1)
- February 2008 (1)
- January 2008 (1)
- November 2007 (1)
- October 2007 (3)
- September 2007 (1)
- August 2007 (2)
- July 2007 (1)
- June 2007 (1)
- May 2007 (8)
- April 2007 (13)
- March 2007 (14)
- February 2007 (6)
- January 2007 (7)

- Recent Comments
  - Weibing on "All methods in Python are effectively virtual"
  - mojtaba on Programming Challenges: Australian Voting
  - pschaeff on Release: Bro 2.3.1-2 on OpenWRT
  - Nam on Release: Bro 2.3.1-2 on OpenWRT
  - Affan on Release: Bro 2.3.1-2 on OpenWRT

- Search
  - Search

- Subscribe
  - Email
    Subscribe

- Meta
  - Log in
  - Entries RSS
  - Comments RSS
  - WordPress.org

- Extra
  - **Total Stats**
    - **217** Posts

- **544** Tags
- **558** Comments
- **266** 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:

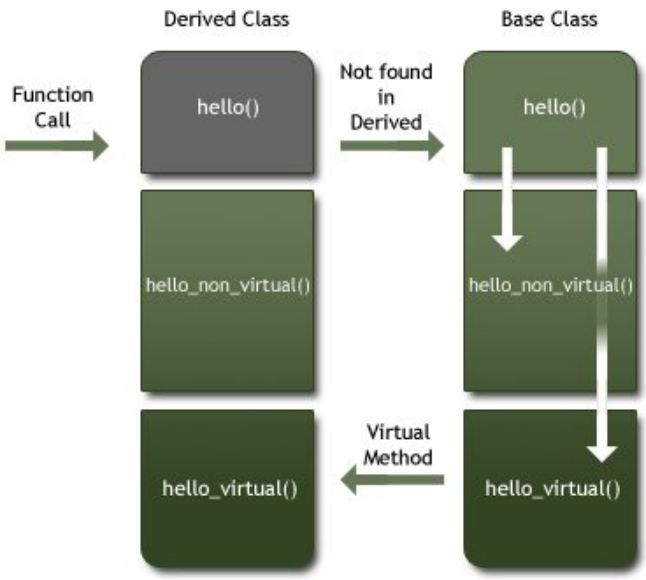
- `hello()`
- `hello_non_virtual()`
- `hello_virtual()`

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

        void hello_virtual()
        {
                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 (4)

4 Comments »

1. Thoroughly enjoyed your post. Very informative!

   Comment by Sheharbano — April 5, 2012 @ 4:29 pm

2. Thanks!!
   really helpful post!!

   Comment by Lokesh — August 17, 2014 @ 11:09 pm

3. Really nice way to explain….

   Comment by Swarup Mallick — April 6, 2015 @ 10:45 pm

4. Really like your explanation.

   Comment by Weibing — August 5, 2017 @ 8:19 am

RSS feed for comments on this post. TrackBack URL

Leave a comment

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

Notify me of followup comments via e-mail

One small verification for man, one giant PITA for bots:

2 × eight =

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License
*Powered by **WordPress***