

What exactly is metaprogramming?

Asked 15 years, 10 months ago Modified 1 year, 7 months ago

Viewed 116k times



207



I was reading an article on TheServerSide on [poyglot programming on the Java platform](#). Some comments in the article refer to metaprogramming as the ability to generate code (perhaps on the fly).

Is metaprogramming the ability to generate code on the fly or is it the ability to inject methods and attributes into existing objects at runtime (like what some dynamic languages like Python, Ruby, and Groovy allow).

metaprogramming

Share

Improve this question

Follow

edited Sep 26, 2016 at 11:37



nbro

16k ● 34 ● 119 ● 212

asked Feb 5, 2009 at 5:05



Parag

12.4k ● 16 ● 60 ● 76

10 You might be interested in this answer
[stackoverflow.com/questions/2565572/...](#) – ewernli Apr 10, 2012 at 9:01

1 @ewernli: That answer is actually better than any of the answers here! – [J D](#) Feb 14, 2017 at 7:12

8 Answers

Sorted by:

Highest score (default)



Metaprogramming refers to a variety of ways a program has knowledge of itself or can manipulate itself.

148



In languages like C#, reflection is a form of metaprogramming since the program can examine information about itself. For example returning a list of all the properties of an object.



In languages like ActionScript, you can evaluate functions at runtime to create new programs such as `eval("x" + i)`. `DoSomething()` would affect an object called `x1` when `i` is 1 and `x2` when `i` is 2.



Finally, another common form of metaprogramming is when the program can change itself in non-trivial fashions. LISP is well known for this and is something Paul Graham championed about a decade ago. I'll have to look up some of his specific essays. But the idea is that the program would change another part of the program based on its state. This allows a level of flexibility to make decisions at runtime that is very difficult in most popular languages today.

It is also worth noting that back in the good ol' days of programming in straight assembly, programs that altered

themselves at runtime were necessary and very commonplace.

From Paul Graham's essay ["What Made Lisp Different"](#):

Many languages have something called a macro. But Lisp macros are unique. And believe it or not, what they do is related to the parentheses. The designers of Lisp didn't put all those parentheses in the language just to be different. To the Blub programmer, Lisp code looks weird. But those parentheses are there for a reason. They are the outward evidence of a fundamental difference between Lisp and other languages.

Lisp code is made out of Lisp data objects. And not in the trivial sense that the source files contain characters, and strings are one of the data types supported by the language. Lisp code, after it's read by the parser, is made of data structures that you can traverse.

If you understand how compilers work, what's really going on is not so much that Lisp has a strange syntax as that Lisp has no syntax. You write programs in the parse trees that get generated within the compiler when other languages are parsed. But these parse trees are fully accessible to your programs. You can write programs that manipulate them. In Lisp, these

programs are called macros. They are programs that write programs.

Programs that write programs? When would you ever want to do that? Not very often, if you think in Cobol. All the time, if you think in Lisp. It would be convenient here if I could give an example of a powerful macro, and say there! how about that? But if I did, it would just look like gibberish to someone who didn't know Lisp; there isn't room here to explain everything you'd need to know to understand what it meant. In [Ansi Common Lisp](#) I tried to move things along as fast as I could, and even so I didn't get to macros until page 160.

But I think I can give a kind of argument that might be convincing. The source code of the Viaweb editor was probably about 20-25% macros. Macros are harder to write than ordinary Lisp functions, and it's considered to be bad style to use them when they're not necessary. So every macro in that code is there because it has to be. What that means is that at least 20-25% of the code in this program is doing things that you can't easily do in any other language. However skeptical the Blub programmer might be about my claims for the mysterious powers of Lisp, this ought to make him curious. We weren't writing this code for our own amusement. We were a tiny startup, programming as hard as we could in

order to put technical barriers between us and our competitors.

A suspicious person might begin to wonder if there was some correlation here. A big chunk of our code was doing things that are very hard to do in other languages. The resulting software did things our competitors' software couldn't do. Maybe there was some kind of connection. I encourage you to follow that thread. There may be more to that old man hobbling along on his crutches than meets the eye.

Share Improve this answer

Follow

edited Dec 12, 2019 at 18:59



Aykhan Hagverdili

29.8k ● 6 ● 48 ● 103

answered Feb 5, 2009 at 5:27



DavGarcia

18.8k ● 14 ● 61 ● 97

15 Don't forget about template metaprogramming in C++. The ability to execute expressions and make decisions at compile-time, and have the results be compiled statically into the final executable. – [Remy Lebeau](#) May 23, 2014 at 3:07

3 I was shocked by in order to put technical barriers between us and our competitors and this is tamade correct. – [Evan Hu](#) Jan 19, 2017 at 9:37

10 Programs that manipulate themselves are a subset of all metaprograms. Metaprogramming in general just means programs that manipulate programs. – [J D](#) Feb 14, 2017 at 7:10

- 1 I agree with @J D, this answer is kind of incorrect, as metaprogramming is programs that manipulate other programs, not just manipulate themselves. You can look at this answer instead:
stackoverflow.com/a/42220709/12097913 – crxyz Feb 13, 2021 at 20:02
-



127

Great question. I'm very sorry to see that none of the answers currently really answer your question correctly. Perhaps I can help...



The definition of metaprogramming is really quite simple: it means programs that manipulate programs.



Your accepted answer says programs that manipulate themselves. Those are indeed metaprograms but they are a subset of all metaprograms.

All:

- Parsers
- Domain specific languages (DSLs)
- Embedded domain specific languages (EDSLs)
- Compilers
- Interpreters
- Term rewriters
- Theorem provers

are metaprograms. So the [GCC compiler](#) is a metaprogram, the [CPython interpreter](#) is a metaprogram, the [Mathematica computer algebra system](#) is a metaprogram, the [Coq theorem prover](#) is a metaprogram and so on.

Other answers have asserted that metaprograms are programs that generate other programs. Those are indeed metaprograms but, again, they are a subset of all metaprograms. The [Fastest Fourier Transform in the West](#) (FFTW) library is an example of such a metaprogram. The source code is written mostly in [OCaml](#) and it generates bits of C code (called codelets) that are combined to create high performance [Fast Fourier Transform](#) routines optimised for specific machines. That library is actually used to provide the FFT routines in Matlab. People have been writing programs to generate numerical methods for decades, since the early days of [FORTRAN](#).

The first programming language that integrated support for metaprogramming was LISP Processor (LISP) language in late 1950s. [LISP 1.5](#) included a number of features that made metaprogramming easier. Firstly, LISP's core data type is nested lists, i.e. trees like `(a (b c) d)`, which means any LISP code can be expressed natively as a data structure. This is known as homoiconicity. Secondly, LISP code can be converted into data easily using QUOTE. For example `(+ 1 2 3)` adds 1+2+3 and `(QUOTE (+ 1 2 3))` creates an expression that adds 1+2+3 when evaluated. Thirdly, LISP provided

a meta-circular evaluator that allows you to use the host interpreter or compiler to evaluate LISP code at run-time, including run-time generated LISP code. LISP's descendants include [Scheme](#) and [Clojure](#). In all of these languages metaprogramming is most commonly seen in the form of programs that modify themselves, typically using macros.

In the 1970s, Robin Milner developed a [MetaLanguage](#) (ML) that evolved into the ML family of programming languages which includes [Standard ML](#) and [OCaml](#) and strongly influenced [Haskell](#) and [F#](#). These languages make it easy to express other languages. In these languages metaprograms are most commonly seen in the form of lexers, parsers, interpreters and compilers.

In 1994, [Erwin Unruh discovered that the C++ template system was Turing complete and could be used to execute arbitrary programs at compile time](#). C++ template metaprogramming brought metaprogramming to the unwashed masses who (ab)used it for many different things including generating numerical methods in the [Blitz++ library](#).

Share Improve this answer


Follow

answered Feb 14, 2017 at 7:59



J D

48.6k ● 14 ● 174 ● 277

-
- 2 Great answer; and the one I was looking for. I was worried that the accepted answer only discussed *reflective* programs, when meta-programming is much more than that. – [afsantos](#)
Dec 5, 2020 at 13:56 
-



Well, metaprogramming is just programming, but it's basically *"writing code that writes code"*.

44



The ability that you mention, when a program can observe and modify its own structure and behaviour is called reflection and it's a type of metaprogramming.



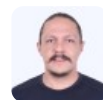
Dynamically typed languages, have powerful runtime reflection features, made possible by the interpreted nature of these languages...

Static typed languages also have powerful metaprogramming techniques, for example the C++ [template metaprogramming](#)...

Share Improve this answer

answered Feb 5, 2009 at 5:33

Follow



[Christian C. Salvadó](#)

826k ● 184 ● 927 ● 843



This is just my personal opinion, which is probably the most liberal definition of metaprogramming.

19

I think it includes:





1. Compile code generation or Runtime code generation (or both)
2. Aspect-Oriented Thinking or Aspect Oriented Programming
3. [DRY](#) Thinking

I think you can get there by using any of these and in combination:

1. Reflection
2. DSLs (Domain Specific Languages)
3. Attributes (.NET) or Annotations (Java)
4. Generics (.NET/Java)
5. Templates (C++)
6. `method_missing` (Ruby)
7. closures / first class functions / delegates
8. AOP - Aspect Oriented Programming

Share Improve this answer

Follow

edited May 23, 2014 at 2:37



Jamal

771 ● 7 ● 22 ● 32

answered Feb 5, 2009 at 5:42



BuddyJoe

71.1k ● 115 ● 301 ● 473

-
- 1 very concise and thoughtful answer. gave me a good menu of things to investigate. thank you! – [swyx](#) Apr 20, 2018 at 17:26
-



9



Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at runtime that would otherwise be done at compile time. In many cases, this allows programmers to get more done in the same amount of time as they would take to write all the code manually, or it gives programs greater flexibility to efficiently handle new situations without recompilation. ([Source](#).)

Basically, it's writing code that outputs more code, which is run to accomplish some goal. This is usually done either within the same language (using javascript to create a javascript string, then `eval` it) or to emit another language (using .NET to create a windows batch file).

Share Improve this answer

answered Feb 5, 2009 at 5:21

Follow



[EndangeredMassa](#)

17.5k ● 8 ● 56 ● 80



7



Metaprogramming is writing a program which outputs another program. This is something languages like Lisp are really good at. It is much easier to do in a language that supports real macros (not C++ macros, but rather ones that can manipulate the code they output) such as Ruby, Lisp, Scheme, etc. than in a language like Java.



One implementation is to create a "domain specific language" which is a way of enhancing a programming language to accomplish a specific task. It can be incredibly powerful if done correctly. Ruby on Rails is a good example of this sort of programming.

If you interested in exploring this method, check out the [Structure and Interpretation of Computer Programs](#) which is one of the seminal books covering the subject.

Share Improve this answer

answered Feb 5, 2009 at 5:27

Follow



Steve Rowe

19.4k ● 9 ● 53 ● 82

-
- 1 Old post, I know, but Metaprogramming is writing a program which outputs another program . Isn't that true for every program not written in binary? You're using the language along with the compiler to output another program (the executable)? – Elliott Aug 20, 2021 at 6:14
-



5



[wikipedia](#) has a nice article on the topic. One does not have to do runtime modifications for something to qualify as metaprogramming. For example, many people use C++ templates to do metaprogramming at compile time.

Share Improve this answer

answered Feb 5, 2009 at 5:21

Follow



Mr Fooz

112k ● 7 ● 76 ● 103



Let's understand this using simple example!



1



```
template<class T>
class Item{
private:
    std::string name;
    T value;
public:
    Item(std::string name, T value)
        : name{name}, value{value} {}
    std::string get_name() const {return name;}
    T get_value() const {return value;}
};
```

In this example, T could be value of any type. For example, we expect integers or doubles, in this case. Now, this will compile but it will not generate any code. It is simply a **blueprint**. The code will be generated by the compiler by using template or blueprint when user uses a **specialized** version of the template or blueprint. That's what meta-programming is all about!

Share Improve this answer

answered Mar 13, 2021 at 0:38

Follow



Saquib Ahsan

61 ● 1 ● 4

Two points: This code is in `c++`, yet this question is not specifically addressed at any language. Secondly, I think that your statement that meta-programming is simply using blueprints really is too narrow. Given that you're familiar with TMP in `C++`, [here's a link](#) to some generic code I wrote to choose types from any variadic at run-time. The important part is I don't see how that has anything to do with blueprints. Also, we could use the same logic to say that functions are blueprints, so we're all always meta-programming. – Elliott Aug 20, 2021 at 6:25
