Multiparadigm Programming with Python Chapter 8

H. Conrad Cunningham

05 April 2022

Contents

Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

Metaprogramming

Chapter Introduction

TODO: Introduction and other missing parts. These are updated consistently for use with other chapters.

Note: In this book, we use the term Python to mean Python 3. The various examples use Python 3.7 or later.

Definition

Basically, *metaprogramming* is writing code that writes code [9,10].

Metaprogramming: the writing of computer programs that can treat computer programs as their data. A program can read, generate, analyze, and/or transform other programs, or even modify itself while running)

We often do metaprogramming in our routine programming tasks but do not call it that.

- Our web applications may generate HTML, JavaScript, and CSS code to enable development of a browser-based user interface.
- Our Java programs may use `instanceof` to check the type of objects or otherwise manipulate itself with the Java reflection package.
- Our C programs may use macros to define new features in terms of existing features [8].
- The PIC little language processor takes a program expressed in an external textual language that describes a picture and generates output expressed in another language that gives instructions to a display program [6,7].

Under the above definition, much of our study of domain-specific languages uses metaprogramming.

Reflexive Metaprogramming

TODO: Decide how to consistently refer to my DSL course examples. Should be consistent with the separate longer Domain-Specific Languages document.

The internal Survey DSL and Lair Configuration DSL are examples of *reflexive (or reflective) metaprogramming* [9,10].

Reflexive metaprogramming: the writing of computer programs that manipulate themselves as data.

This manipulation may be at compile time, involving a phase of transformations in the code before the final program is generated. Or it may be at runtime, involving manipulation of the program's metamodel or generation of new code that is dynamically executed within the program.

The Survey DSL is a Ruby internal DSL. It takes advantage of Ruby's metaprogramming facilities such as the abilities to trap calls to undefined methods, add methods or variables dynamically to existing objects at runtime, and execute dynamically generated strings as Ruby code. It also uses Ruby's first-class functions (closures) and flexible syntax – although these are not technically metaprogramming features of Ruby.

The Lair Configuration DSL programs use the metaprogramming features of Lua and Python in similar ways.

Consider relatively common languages and their metaprogramming features.

- Java is a statically typed, compiled language. What are metaprogramming features available in Java?

  It has dynamic class loaders, a reflection API, annotation processing, dynamic method invocation (a JVM feature), JVM bytecode manipulation (mostly with external tools), etc. Java 8+ also has first-class functions and other features useful in metaprogramming.

- Lua is a dynamically typed, interpreted language. What are the metaprogramming features available in Lua?

  It has metatables, metamethods, manipulation of environments, a debug library (introspection/reflection features), `loadfile` and `loadstring` functions to dynamically execute code, extensions in C, etc.

What about Python?

The reflexive metaprogramming features of Python 3.7 and beyond is the primary topic of this set of lecture notes.

Why Study Reflexive Metaprogramming?

In everyday application programming, we often use the products developed by metaprogrammers, but we seldom use the techniques directly.

In everyday programming, use of reflexive metaprogramming techniques should not be one of our first approaches to a problem. We first should explore techniques supported by core language, its standard libraries, and stable extension packages.

If no acceptable solution can be found, then we can consider solutions that use reflexive metaprogramming techniques. We should approach metaprogramming with great care because these techniques can make programs difficult to understand and can introduce vulnerabilities into our programs. We should design, implement, test, and document the programs rigorously.

However, reflexive metaprogramming can be an important tool in a master programmer's toolbox. If our jobs are to develop software frameworks, libraries, APIs, or domain-specific languages, we can use these techniques and features to develop powerful products that hide the complexity from the application programmer.

Even when our jobs are primarily application programming, understanding reflexive metaprogramming techniques can improve our abilities to use software frameworks, libraries, and APIs effectively.

Reflexive Metaprogramming in Python

TODO: Update this to better reflect, what the final notes cover and include forward references as appropriate.

The reflexive metaprogramming features of Python include:

1. Decorators
2. Metaclasses
3. Descriptors
4. Import hooks
5. Context managers
6. Annotations (e.g., type hints)
7. Abstract Syntax Tree (AST) manipulation
8. Frame hacks
9. Execution of strings as Python 3 code (`exec`, `eval`)
10. *Monkeypatching* (i.e., direct dynamic manipulation of attributes and methods at runtime)

We have already used the final two in our implementation of domain-specific languages. We will look at some of the others in these notes. In particular, Chapter 9 looks at use of decorators and metaclasses.

What Next?

TODO

Chapter Source Code

TODO

Exercises

TODO: Decide if any are appropriate.

Acknowledgements

I developed these notes in Spring 2018 for use in CSci 658 Software Language Engineering [3]. The Spring 2018 version used Python 3.6.

Teaching a special topics course on "Ruby and Software Development" in Fall 2006 kindled my interests in domain-specific languages and metaprogramming.

Building on these interests, I taught another special topics course on "Software Language Engineering" in which I focused on Martin Fowler's work on domain-specific languages [5]; I subsequently formalized this as CSci 658. (I have collected some overall ideas on in my notes on Domain Specific Languages [4].)

The overall set of notes on Python 3 Reflexive Metaprogramming is inspired by David Beazley's Python 3 Metaprogramming tutorial from PyCon'2013 [1]. In particular, some chapters adapt Beazley's examples. Beazley's tutorial draws on material from his and Brian K. Jones' book *Python Cookbook* [2].

I retired from the full-time faculty in May 2019. As one of my post-retirement projects, I am continuing work on this textbook. In January 2022, I began refining the existing content, integrating (e.g., using CSS), constructing a unified bibliography (e.g., using citeproc), and improving the build workflow and use of Pandoc.

I maintain these notes as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the notes to HTML, PDF, and other forms as needed.

Terms and Concepts

TODO: Update

Metaprogramming, reflexive metaprogramming.

References

[1] David Beazley. 2013. Python 3 metaprogramming (tutorial). Retrieved from http://www.dabeaz.com/py3meta/

[2] David Beazley and Brian K. Jones. 2013. *Python cookbook* (Third ed.). O'Reilly Media, Sebastopol, California, USA.

[3] H. Conrad Cunningham. 2018. Python 3 reflexive metaprogramming. University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/csci658/notes/PythonMetaprogramming/Py3RefMeta.html

[4] H. Conrad Cunningham. 2022. *Domain-specific languages.* University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/docs/DSLs/DomainSpecificLanguages.html

[5] Martin Fowler and Rebecca Parsons. 2010. *Domain specific languages.* Addison-Wesley, Boston, Massachusetts, USA.

[6] Brian W. Kernighan. 1984. *PIC—a graphics language for typesetting, revised user manual.* Bell Laboratories, Computing Science, Murray Hill, New Jersey, USA. Retrieved from http://doc.cat-v.org/unix/v8/picmemo.pdf

[7] Eric S. Raymond. 1995. Making pictures with GNU PIC. Retrieved from https://lists.gnu.org/r/groff/2011-08/pdfbLauVhlfQs.pd

[8] Richard M. Stallman and Zachary Weinberg. 2022. The c preprocessor. Retrieved from https://gcc.gnu.org/onlinedocs/cpp/

[9] Wikpedia: The Free Encyclopedia. 2022. Metaprogramming. Retrieved from https://en.wikipedia.org/wiki/Metaprogramming

[10] Wikpedia: The Free Encyclopedia. 2022. Reflective programming. Retrieved from https://en.wikipedia.org/wiki/Reflective_programming