PyHEP Workshop, in conjunction with CHEP 2018
Additional note

% hep
HEP 4.2.0 (#29, 27 Jan 1994, 14:27:38) [GCC 2.5.7] on sun4-solaris
>>> import python
bailing out at line 1
#
%

Jeff Templon[1]*

Abstract
Python took a while to become an accepted language in High-Energy Physics. This short paper traces some of the history of this path to acceptance, and suggests some reasons for this apparently-slow uptake.

Keywords
High-Energy Physics — Python — Computing

[1] *Nikhef, Science Park 105, 1098 XG Amsterdam, The Netherlands*
***Corresponding author**: templon@nikhef.nl

Contents

Introduction 1
1 Before Python 1
2 My first four months with Python 1
3 Resistance to Python 2
4 Python into the mainstream 3
5 Reflections 3
5.1 What Gap Was Python Filling? 3
5.2 Why Did Python Fill This Gap? 3
6 Conclusions 4
References 4

Introduction

This paper contains some personal notes about the "early days" of Python in High-Energy and Nuclear Physics, along with some thoughts about why Python gained acceptance and why the path to acceptance went like it went.

1. Before Python

I did my Ph.D. at Indiana (IUCF) in Experimental Nuclear Physics. The main computing environment was VAX/VMS, and on that platform, FORTRAN absolutely rocked. "Scripting" things except for canned command sequences was not common, although the DEC command language DCL could be used to do that. Halfway through my time there, the lab purchased a license for SPEAKEASY[1] (which still exists!). The program was an interpreter containing a large library of math, statistics, and graphics functions, and you could define your own functions as well. I used this often as a sort of desk calculator, usually reserving FORTRAN codes for things that needed to do more complex I/O, or for calculations that took too long in SPEAKEASY.

[1] http://www.speakeasy.com/default.htm

At some point I discovered GNU `awk`, which had been ported to VAX/VMS system. After that, most of my analyses were scripted. The Fortran programs were written to Do One Thing Well; the orchestration, data gathering and tabulation of all the hundreds of Fortran runs were done by `awk` scripts.

Like most graduate students, I found ways to avoid writing my dissertation. One of them was playing with exotic little languages, and one of them was called ABC[2], a teaching/prototyping langauge from the CWI in Amsterdam. It was a fun little language, very easy to learn.

Upon graduation I moved to NIKHEF-K in Amsterdam for my first postdoctoral position. There were no VAXen, so no DEC FORTRAN, no VMS, no SPEAKEASY. Instead there were Sun machines, Solaris-flavored Unix, FORTRAN-77, and `bc` and `dc` as desk calculators. I got used to Unix and standard FORTRAN-77 fairly quickly, but I missed the interactive calculator and statistics library of SPEAKEASY. `bc` and `dc` didn't work at all for me, leading to my (mis)use of `awk` for a lot of things. I write "misuse" since the `awk` paradigm is "transform input to output", but I was using it for other things too; many of my `awk` scripts had all the action in the `END` clause.

2. My first four months with Python

On 27 January 1994, the announcement of Python 1.0.0 came out on `comp.lang.misc`. I decided to download it and check it out because a) it was available on Solaris, b) it was an interpreted language claiming to come with a large library of standard modules (so a potential SPEAKEASY replacement), and c) it came from the same place as that little ABC language and I had liked that one, maybe this one was okay too.

I have a reputation of being the first guy in our field to use Python. I'm not — Jon Eisenberg at the University of

[2] https://en.wikipedia.org/wiki/ABC_(programming_language)

Washington reported in October 1993 that he was "developing a data analysis application using python as the interface and user programming language." I could not discover whether he completed it. David Williams (an ex-colleague from IUCF) also made a few posts on the Python mailing list a couple of days before I did.

My first substantive post to the Python mailing list was a request for comments on a little program I'd written while playing around with the language – Friday afternoon of March 4, 1994. The exchange was a lot of fun, with both Guido van Rossum and Tim Peters[3] making most of the comments. At the end of my last message in that thread, I made a joke that became reality:

> I don't think I'll become a class writer just yet. Maybe next year. Although I have this nagging temptation to write a good data analysis program using Python ... all we have here is PAW which is a major pain[4].

That was 9 March 1994. Twelve days later:

> I did write that data-analysis project[5] . . . what I wrote was a preprocessor; it takes as input some files which specify how you would like the data to be analyzed, and translates this information into FORTRAN . . . a FORTRAN main program . . . includes the Python[-generated] FORTRAN output and does just a little bookkeeping. . . .

The code was named DATAN; not that that's important, but useful since I will refer to it later.

After this, things went very fast:

12 april first kinematics program

21 april four-vector class library

22 april run planning program (count rates, kinematics, beam time estimates for needed statistical accuracy) for NIKHEF-K Experiment 93-02[6]

25 may DATAN was used as the online analysis and monitoring package for an experiment at NIKHEF-K

3. Resistance to Python

After this point, I was using Python for almost everything. I went a bit too far; I was using it in areas where `bash` would have been more appropriate. Time-critical stuff was still (mostly) written in Fortran and the things that beg to be written in `awk` (yes, they do exist) were still written in `awk`, the rest was Python.

It took a long time before I met anybody else using Python. For a couple of years, people literally said I was crazy: "How could you use this language for serious work?? YOU DOWNLOADED IT OFF THE INTERNET!!" Recall that in 1994, you could still have made the argument that WWW stood for Wild West Web. When I left Nikhef in 1995, one of my then ex-colleagues rewrote DATAN. The syntax didn't change at all, but it was rewritten in a "proper language" (C).

About this time, the idea of an "extension language" became popular. In this model, large programs would not have a bespoke CLI and syntax; the CLI would be a library linked in with the program. There were quite a few candidates proposed for this library, Tcl being the front runner in the beginning. The "Tcl war" happened in late 1994 - Richard Stallman had written a Usenet post entitled "Why you should not use Tcl" (as an extension language).

In late 1995 the ROOT project had its first public release. ROOT had embraced the extension-language concept from the beginning, albeit with an interesting choice: the extension language was the same as the compliled language, C++. The difference was that the extension language was being interpreted. When the collaboration I was working on at the time started to consider ROOT, I had several long conversations with Fons Rademakers and Rene Brun[7] in this period, about using Python for the CLI instead of CINT. The ROOT team rejected Python firstly on the principle that the extension and main languages should be the same; asking a physicist to learn two languages was considered asking too much. The second reason for rejection was that Python was an "exotic language".

The DØ experiment at Fermilab was, as far as I can tell, the first to use Python as an extension language, as evidenced by the following excerpt from a paper[1] from CHEP 1997.

> DØ has made the decision to move all large software projects to C++. Their framework approach has a set of modules that execute sequentially, each having a specific task. The glue that holds the individual software packages together will be an interpreted script system. The main task of this framework is to "guide" data between the various modules/packages ... prototype framework based on the Python scripting language has been developed and is ready for use.

Working in Nuclear Physics, I didn't know about this work until I visited Fermilab a year later[8] That visit influenced my own thinking, reflected in this excerpt from a technical note[2] presented at a Hall A (Jefferson Laboratory) collaboration workshop in early 1999.

[3]Of "import this" fame (try importing that module from the python command prompt!

[4]Disclaimer: I was used to the histogram/condition table package in the IUCF version of the Los Alamos "Q" system. The Q and PAW paradigms were worlds apart. Hence my frustration with PAW.

[5]Implementing the paradigm and exact syntax of Q.

[6]I still have this program, and tested it while researching this paper in early 2018. It still runs under Python 2.7.15 with a single change; the `class.init()` syntax is different.

[7]Rene and I had one of these conversations at a barbecue joint in southeastern Virginia, close to Jefferson Lab. They had paper tablecloths; we filled our tablecloth with sketches about how physics computing and analysis should work. I had an extremely enjoyable, entertaining, educational and inspiring afternoon, one of the most memorable of my career. But I did not manage to convince Rene.

[8]May 26, 1998, to give a talk about programming languages and tools for physics.

> Most of the code would be written in C or C++, but the integration would be done through Python. This enables the uninitiated to make simple modifications to the analysis which were perhaps not thought of by the authors; all the neophyte needs to know is how the interfaces work. On the other hand, it will force the code authors to make the analysis subsystems independent of each other (one of the big problems with the current code), and will encourage rigorous testing of subunits.

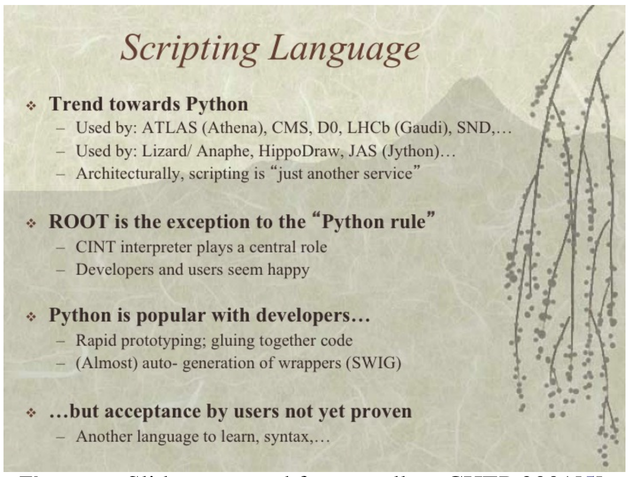
4. Python into the mainstream

I was unable to discover, via CHEP proceedings, how things had progressed in the period immediately following, as the CHEP 1998 site is no longer online, and I could find no proceedings. A search for papers and slides corresponding to talks submitted to that CHEP turned up a few mentioning Python, all from Fermilab.

At the next CHEP (2000), there were two firsts:

1. the first talk mentioning Python in the title, "Dynamic Graphical User Interfaces using XML and JPython"[3] (author based at Fermilab), and
2. the first non-FNAL CHEP reference to Python I could find: "AIDA (LHC++), User Interface in Python"[4], from CERN.

Python seems to have really taken off by the CHEP in 2001, as evidenced by the following excerpt from Philippe Canal's summary talk of the Data Analysis and Visualization track at CHEP 2001 (September) (see fig. 1.)



Scripting Language

- **Trend towards Python**
  - Used by: ATLAS (Athena), CMS, D0, LHCb (Gaudi), SND,…
  - Used by: Lizard/ Anaphe, HippoDraw, JAS (Jython)…
  - Architecturally, scripting is "just another service"
- **ROOT is the exception to the "Python rule"**
  - CINT interpreter plays a central role
  - Developers and users seem happy
- **Python is popular with developers…**
  - Rapid prototyping; gluing together code
  - (Almost) auto- generation of wrappers (SWIG)
- **…but acceptance by users not yet proven**
  - Another language to learn, syntax,…

Figure 1. Slide excerpted from a talk at CHEP 2001[5].

By the next CHEP (March 2003), there was an Athena python talk, and LHCb's entire framework was heavily Python based, with DIRAC, GANGA, Gaudi, and Bender. Python was a mainstream HEP language.

5. Reflections

One question that was raised during discussions about holding the 1st PyHEP workshop is: "why did it take so long" for Python to be adopted in HEP? In my opinion, the answer has several components.

5.1 What Gap Was Python Filling?

Looking back on how I was using first `awk` and later Python, the earlier use case was a kind of orchestration, the later one was being able to rapidly code up the gist of something I was trying to achieve, leaving a lot of the guts to Python libraries and/or external utility programs.

Regarding the first, when I worked at TUNL on the VAX 11/780 there, it was pretty normal for any significant program to run for hours or even overnight. The VAX 8650 at IUCF was ten times faster. I could create a bundle of work large enough to run overnight by, for example, using `awk` to orchestrate hundreds of fitting runs (still good old FORTRAN programs), parse the output to synthesize a second dataset on which to run a sensitivity analysis. Python was great at doing stuff like this.

Regarding the second: when program runs take hours or days, time-to-solution can be significantly improved by a significant initial investment in coding, reducing the run times by choice of programming language and good algorithms and data structures. When these run times are reduced to minutes, development time dominates the time to solution.

RISC and x86 systems became common (and cheap) in the early nineties; in just five years the Dhrystone rating for the x86 series increased by a factor of 20[9]. For me personally, this was a factor of 200 performance increase in ten years.

Physicists were looking for a rapid-development language system.

5.2 Why Did Python Fill This Gap?

All The Right Stuff Python had a lot going for it. The language is simple, readable, and even in the 90s had a large library of useful stuff. There was built-in support both for extending Python via external libraries, and for using Python as an extension language / CLI within a program.

A clear standard Python had, at the beginning, a single implementation. Contributions were encouraged and accepted, however there was a Benevolent Dictator who had the final say in most matters.

The Oberon-2 language provides a useful contrast. I was part of a (very) small group trying to get the Nuclear Physics folk to move to Oberon-2 instead of C++. Our failure was not due the language; much of Oberon-2 wound up being reborn in Golang. The problem was the lack of a reference implementation. The language was defined without mention of a standard library; two universities provided (incompatible) runtimes, neither of which achieved "clear winner" status. Oberon-2 remained a niche, academic language.

The platform revolution In the early 1990s, "supported platforms" were the norm. IUCF computing was overwhelmingly VAX/VMS with a handful of Ultrix systems; when I came to Nikhef in 1992, everything was SunOS. While researching this paper, I found a CERN Computer Newsletter from September 1994, which mentioned CERN

[9] 1989: 8.7 Dhrystone MIPS for an i486DX chip; in 1994, 188 MIPS for the new Pentium chip.

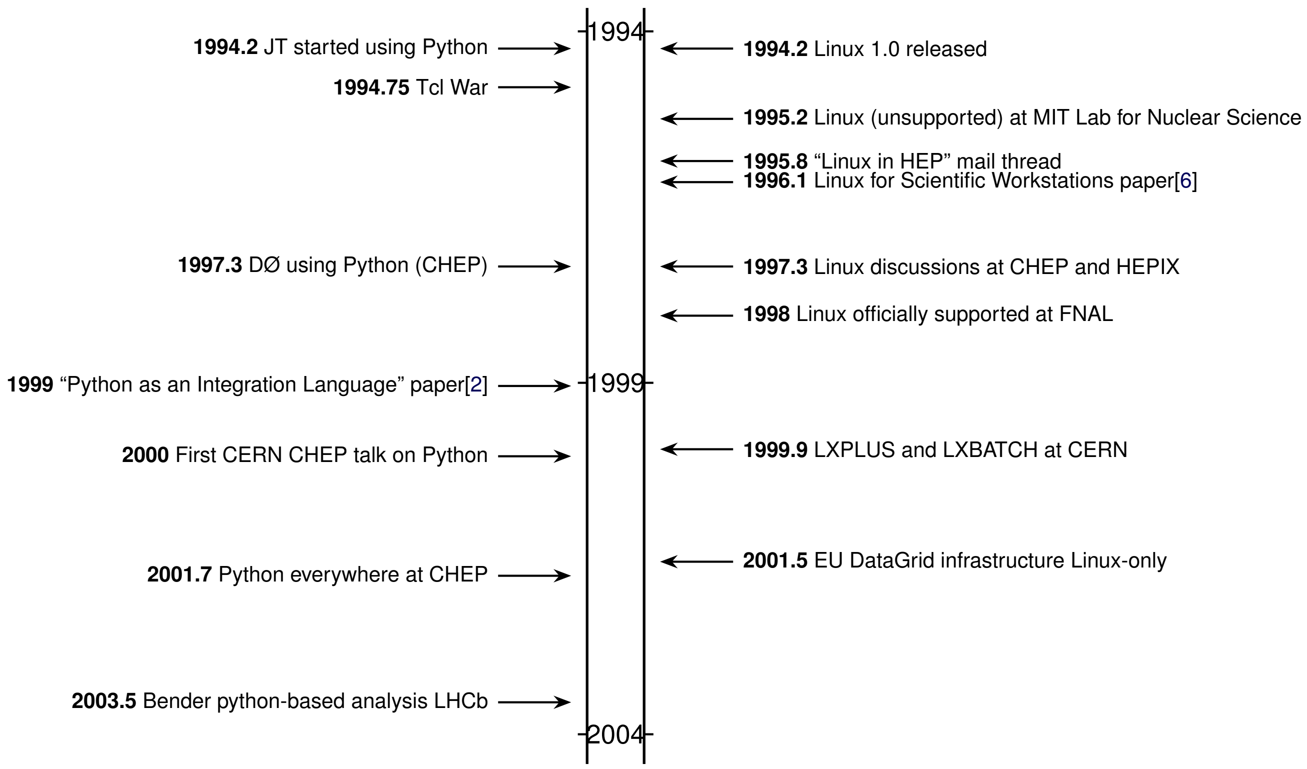

1994
1999
2004
1994.2 JT started using Python
1994.75 Tcl War
1997.3 DØ using Python (CHEP)
1999 “Python as an Integration Language” paper[2]
2000 First CERN CHEP talk on Python
2001.7 Python everywhere at CHEP
2003.5 Bender python-based analysis LHCb
1994.2 Linux 1.0 released
1995.2 Linux (unsupported) at MIT Lab for Nuclear Science
1995.8 “Linux in HEP” mail thread
1996.1 Linux for Scientific Workstations paper[6]
1997.3 Linux discussions at CHEP and HEPIX
1998 Linux officially supported at FNAL
1999.9 LXPLUS and LXBATCH at CERN
2001.5 EU DataGrid infrastructure Linux-only

Figure 2. Python and Linux Timeline. The “Linux in HEP” mail thread included people working at Queen Mary University (London), Neils Bohr Institute (Copenhagen), FNAL, DESY, CERN, George Mason University, and MIT.

running central VM/CMS and VAX/VMS systems, and Sun, DEC, HP, Apollo, and SGI unix workstations. Such systems were shipped with an operating system and were generally supported, maintained, and operated by the computer systems group.

Python was not a “standard” part of any of these systems. “Normal” physicists were used to using the “standard” software that either came with these systems, or was installed on them by the computer group.

This started to change when the Linux movement for HENP started up in the mid nineties. With Linux, the whole dang thing was “downloaded off the internet”. Most Linux distributions had Python installed as a standard package. As Linux became more mainstream, Python became more of a standard package.

It takes a bit of time To the best of my knowledge, Fermilab was the first big lab using Linux. Fermilab people were involved in the early discussions on the linux-hep mailing list, in late 1995. A bit more than a year later, was that first FNAL contribution at CHEP reporting use of Python. CERN was later to the game; in late 1997 Linux was mentioned as an unsupported platform on the ASIS repository; in late 1999 LXPLUS and LXBATCH (the Linux service at CERN) was announced, and at CHEP 2000 there was the first talk from CERN mentioning Python, 18 months later “the trend is towards Python”.

So part of the time-to-uptake was, it seems, moving to a platform where Python was a standard part of the ecosystem; the other part being, once that transition was made, the time it takes for significant software to become visible to the community, for example via presentations at CHEP. It may even be the 18 month cycle of CHEP we’re seeing; possibly the timescale of uptake is more like the months described in my personal experience, but then one has to wait on average 9 months before another CHEP rolls around.

Figure 2 shows a timeline of the ten-years following my first experience with Python, showing some of the Python milestones on the left-hand side, and Linux milestones on the right.

6. Conclusions

Most of this paper is dedicated to telling the story of Python adoption from my personal perspective. In that respect there aren’t really any conclusions. I will say, it was personally a lot of fun and gave me satisfaction to see how abundantly my foray into Python paid off in terms of my research productivity.

For the reflections part, Python seems to have “happened” for our field because it was a well-suited solution to our desire, as a field, to reduce development time, this desire being driven by the rapidly diminishing execution times associated with the tremendous gains in performance (and performance/price ratio) over the nineties.

The interested reader could do a web search for Jim Pivarski’s thoughts on programming languages, he is examining similar issues within a much broader context.

References

[1] S. Lammel. Computing models of major HEP experiments: D0 and CDF. In *Proceedings, 9th International Conference on Computing in High-Energy*

Physics (CHEP 1997): Berlin, Germany, April 7-11, 1997, 1997.

[2] J. A. Templon. Python as an Integration Language. Technical Report SPAG-1998-02, The University of Georgia, Department of Physics and Astronomy, January 1998.

[3] G. Guglielmo. Dynamic Graphical User Interfaces using XML and JPython. In *Computing in High-Energy Physics (CHEP 2000): Padova, Italy, Febrary 7-11, 2000*, 2000.

[4] A. Pfeiffer. Libraries for HEP Computing (LHC++). In *Computing in High-Energy Physics (CHEP 2000): Padova, Italy, Febrary 7-11, 2000*, 2000.

[5] Philippe Canal and Lucas Taylor. CHEP 2001: Data Analysis & Visualization. Computing in High-Energy Physics (CHEP 2001): Beijing, P.R. China, September 3-7, 2001, 2001.

[6] J. A. Templon and P. F. Dubois. Evaluation of PC/Linux Systems for Use as Scientific Workstations. *Computers in Physics*, 10(1):49–55, 1996.