How Python Saves Years of Life Every Day

Neal Norwitz
2019-03-30

Air Traffic Control messaging system

Team: 5-10 people

Goal: create a product to bring SMS to existing Air Traffic Control systems

> Provide ~500 APIs to handle all the encoding/decoding (ie, bit twiddling) of each message (that were defined by various international standards)
>
> eg, CLIMB TO AND MAINTAIN BLOCK [altitude] TO [altitude]
>
> WILCO
>
> Standards require **lots of tedious** bit manipulation.

My first impression

```
>>> import string
>>> name = string.capitalize('neal')
>>> print "Hi, my name is %s" % `name`
Hi, my name is 'Neal'
```

Strings are built in, so why import a module?

WTF is up with those 3 types of quotes?

Print is a statement, ok, kinda cool that it’s built in, but whatever.

Python 2.1 (2001)

Leave failing startup doing Java, take a break from working.

Gave Python a serious try for a personal project.

Wow, it actually worked pretty well. It was very easy to create.

My “break” only lasted about a month before forming startup with friends.

Previous company wanted us to do contract work for a new application.

Let’s use Python to create it.

What happened?

Refactored code from:

```
  import boring_module
  import some_module

  def foo():
-     some_module.bar()
```

And I **forgot to test**.

Refactored code from:

```
  import boring_module
  import some_module

  def foo():
+     better_module.bar()
```

And I **didn't have automated tests**.

Basic Python Introspection

```
>>> def foo(): bar()
...
>>> print type(foo)
<type 'function'>
>>> print dir(foo)
['__call__', '__code__', '__defaults__', '__dict__', '__doc__',
'__globals__', '__module__', '__name__', ...]
>>> import dis
>>> dis.dis(foo)
  1           0 LOAD_GLOBAL              0 (bar)
              3 CALL_FUNCTION            0
              6 POP_TOP
              7 LOAD_CONST               0 (None)
             10 RETURN_VALUE
>>> print 'bar' in foo.__globals__
False
```

Warning types in PyChecker

6 Deprecated (using deprecated APIs)
39 Error (likely errors)
4 Security (using bad APIs)
9 Style (function length, line length, etc)
6 Unused Entities (functions, variables, etc)
39 Warning (possible errors)

Python 2.4 (2005): Google

I got tired of my own business.

Decided to take another break.

Google’s hiring machine makes a mistake and asks me to join. Google likely hired me based on my work on open source and in particular my contributions to Python as a core committer.

Summary: Python Language Rules

We summarize our usage policy for Python language features below.

1. pychecker: Required

Cppclean - approximate C++ parser

Goal:

Parse enough of C++ to identify cleanup opportunities.

Non-goals:

Graceful handling of syntax errors or even compiling C++ code.

Implementation detail:

Use Python.

Generators

From https://wiki.python.org/moin/Generators

Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop.

This means that values can be computed lazily or on demand which can also improve performance of CPU and memory.

See PEP 255 for the original definition.

Approach: Tokenize using generators

```
def get_tokens(source):
  i = 0
  while i < len(source):
    start, i = skip_whitespace(source, i)
    if c.isalpha() or c == '_':
      token_type, i = get_end_of_identifier(source, i)
    elif c == '"':
      token_type, i = get_end_of_quote(source, i)
    elif ...:
      token_type, i = ...
    # ~10 total cases not shown here ...
    yield token_type, source[start:i], start, i
    # ...
```

Python 2.7 (2012)

I got tired of C++ at Google (after doing it for 4-5 years).

Joined YouTube to do Python again.

I started on the search frontend team. Spent time ramping up on the YouTube code base. Needed to figure out what to work on. Google likes fast responses. I like fast responses.

YouTube cares about latency.

Can I improve latency?

YouTube style code (vastly simplified)

```
def handle_http_request(req):
  # Do lots of work common to all requests.
  if is_search_request(req):
    html = handle_search()
  # ...
  send_http_response(html)

def handle_search():
  # Talk to backends, get data, process, etc.
  return generate_search_html()
```

HTTP Chunked Transfers

YouTube sent everything in one block. HTTP can send results in chunks.

Servers can send CSS early while waiting for backend responses.

Allows browsers to be ready to layout and display results when they come in.

ie, the browser can do work in parallel with YouTube servers.

YouTube style code implementing HTTP chunking

```
def handle_http_request(req):
  # Do lots of work common to all requests.
  if is_search_request(req):
    response = handle_search()
  # …
  if isinstance(response, str):
    send_http_response(response)
  else:
    for chunk in response:
      send_http_response_chunk(chunk)

def handle_search():
  yield generate_search_html_header()
  # Talk to backends, get data, process, etc.
  yield generate_search_html_body()
```

Comparison of old vs new code

```
Infrastructure:
  send_http_response(html)
```

```
Infrastructure:
+ if isinstance(response, str):
    send_http_response(response)
+ else:
+   for chunk in response:
+     send_http_response_chunk(chunk)
```

```
Caller:
  def handle_search():
-   return generate_search_html()
```

```
Caller:
  def handle_search():
+   yield generate_search_html_header()
+   yield generate_search_html_body()
```

Note: **no existing code** needed to change other than the driver. ie, safer to implement **and** less effort.

Work, work, work

I investigate the problem, from the bug: “I am able to reproduce this exception. I don't know how it occurs, but I think I can code around it.”

From the checkin: “Ensure that response_buffer is always closed. This will help prevent dangling generators from hanging around.”

The fix was to add the close of the generator in the finally clause:

```
+   try:
      for chunk in response:
        send_http_response_chunk(chunk)
+   finally:
+     response.close()
```

Bug #2

A teammate says: “The outermost loop in our stack of generators is not closing the generator correctly. It should have a finally block that calls self.result.close()”

See PEP 342 (Coroutines via Enhanced Generators) for details.

Damn. I fixed the problem in front of me, but didn’t think to investigate the entire stack. I fixed a **symptom**, but not the **root cause**.

Close the generator in this code too. Investigate for more possible problems.

Start homepage experiment for the third time.

Results

~630,720,000,000 +/-

Results

~630,720,000,000 +/-

Web page latency is usually measured in milliseconds.

HTTP chunking on the 3 YouTube high traffic pages saved billions of milliseconds every day (ie, years).

People watch more YouTube.

Lots of metrics were better. I choose to believe that people were happier. :-)

Value is all that matters

Value is the difference between costs and benefits. Focus on **value**. Maximize **value**. Get the *biggest wins* with the *least effort*.

Don’t worry about the small things. Don’t worry about what **seems impossible**. Determine what is or isn’t possible *for yourself based on data*.

Know what *is **and** isn’t* important. Scope your solution to the **minimum required**.

Focus on the next steps, know where you want to be going. Make sure you are working towards the future you are trying to achieve.

Ensure you have **reasonable and high expectations** and they are in sync with others.

More general principles

Challenge **all assumptions and prejudices**, especially your own.

Use your time wisely, it's a *precious and limited* resource. Use your time to learn. Never stop learning.

Always make things better, do simple things to reduce technical debt, when it adds value. eg, tests are like insurance, you hope you don't need them but at some point you will. You won't know which will be important. Do enough just testing to get the **best value**.

Push yourself hard. Try. Experiment. You **will make mistakes**. The mistakes you make don't define you. How you **handle** them is what matters.