The Bakery

How PEP 810 Sped Up My Bread Operations Business

Jacob Coffee

Python Software Foundation Staff | Litestar Maintainer

What We'll Cover

1. **The Problem**
   Why Python startup matters
2. **PEP 810 Explained**
   Explicit lazy imports syntax
3. **Live Demo**
   breadctl in action
4. **Real-World Impact**
   Meta, HRT, and beyond
5. **Use Cases**
   Beyond CLI tools
6. **Migration Guide**
   How to adopt in your projects

The Problem

Python startup is expensive

```
$ time uv run breadctl --help
Usage: breadctl [OPTIONS] COMMAND
Commands: bake, deliver, inventory

real    0m0.234s  ⚠ 234ms just to show help!
```

Eager Loading by Default

Python imports ALL modules at startup

Memory Bloat

Loaded modules stay resident even if unused

Cold Start Tax

Serverless, workers, tests all pay this cost

TYPE_CHECKING Workarounds

Ugly hacks to avoid runtime import costs

The cost is real:

Meta observed 50-70% startup reduction potential • 30-40% memory savings • PySide 35% startup improvement

What is PEP 810?

Explicit Lazy Imports for Python

> *Import a module only when it's first accessed, not when the import statement is executed.*

50-70% Faster

Startup time reduction observed

30-40% Less Memory

Modules not loaded = memory not used

Explicit Syntax

lazy keyword - no surprises or global flags

PEP 810 was accepted!

Coming to Python 3.15

The Syntax

Side-by-side: Normal vs Lazy imports

normal.py

```
# All loaded at startup
from breadctl import bake
from breadctl import deliver
from breadctl import inventory

# bake: collections, itertools
# deliver: httpx
# inventory: sqlite3

# --help loads ALL!
```

lazy.py

```
# Loaded only when accessed
lazy import breadctl.bake as bake
lazy import breadctl.deliver as deliv
lazy import breadctl.inventory as inv

# Nothing loaded yet...

bake.run()   # NOW it loads

# --help? Fast! ⚡
```

How It Works

Step 1 of 3

1

Parse Phase

lazy import foo creates a LazyModule proxy in the namespace

```
lazy import httpx  # Creates proxy, no import yet
```

How It Works

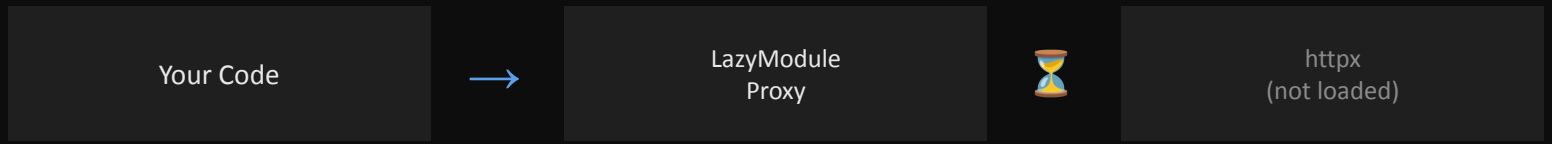
Step 2 of 3

2

Wait Phase

Module is NOT imported yet - proxy sits dormant, no I/O or execution

```
# httpx proxy exists but module not loaded
# No network code, no dependencies
```



How It Works

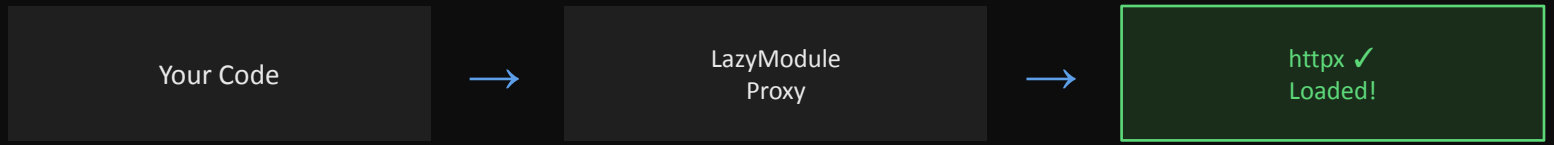
Step 3 of 3

3

Access Phase

First access to foo.bar triggers actual import - proxy replaced transparently

```
response = httpx.get(url)  # NOW httpx loads!
```



How It Works

The complete picture

1. **Parse**
   Create proxy
2. **Wait**
   Proxy dormant
3. **Access**
   Real import

Key Insight

If a command is never run, its imports are never loaded.

--help stays fast because it doesn't touch the heavy modules!

Tab completion, subcommand routing, help text - all fast.

Live Demo 👻

breadctl in action (scary live demo time!)

Normal (eager imports)

```
$ time uv run breadctl --help
real 0m0.234s
```

Lazy (PEP 810)

```
$ time uv run breadctl-lazy --help
real 0m0.164s
```

Demo Commands

```
uv run breadctl --help
uv run breadctl-lazy --help
uv run breadctl bake
uv run breadctl-lazy bake
make bench
# Show the charts!
```

```
# Deep dive: see what's loading
$ python -X importtime -c "import breadctl_cappa.normal" 2>&1 | head
$ python -X importtime -c "import breadctl_cappa.lazy" 2>&1 | head
```

Benchmark Results

Real numbers from breadctl (Cappa framework)

--help command

23% faster

215ms → 165ms

Module import time

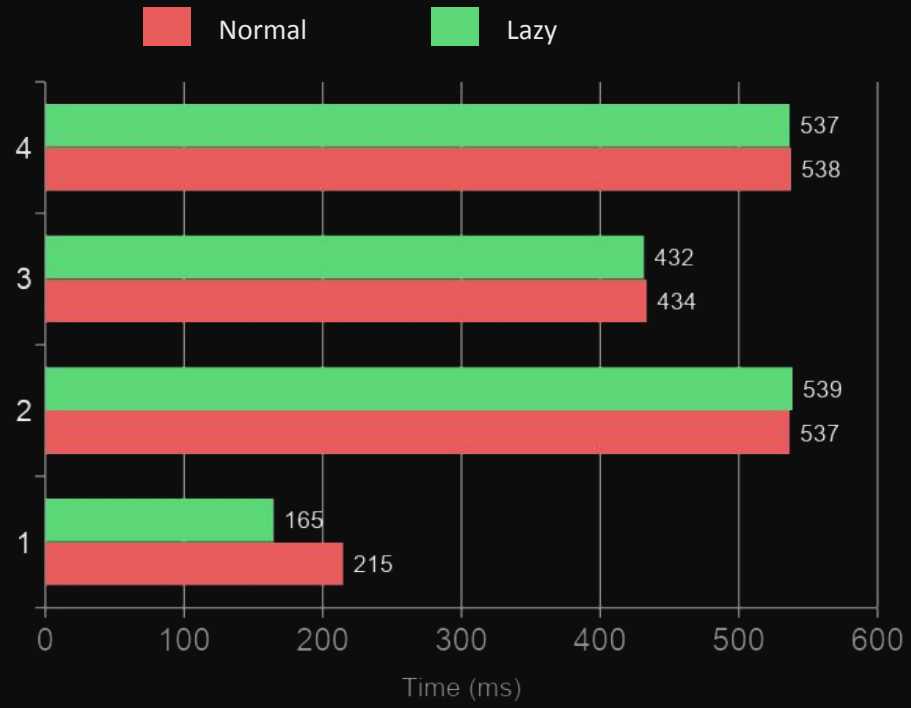
26% faster

198ms → 147ms

inventory command

~same

538ms (needs all modules anyway)



Real-World Impact

Production implementations proving the concept

Meta (Cinder)

Instagram's Python fork with lazy imports

70% startup reduction

40% memory savings

Hudson River Trading

Custom Python fork for trading systems

Module-level lazy imports in production for years

PySide (Qt for Python)

Official Qt Python bindings

35% startup improvement

💡 Key Insight

These aren't experiments - they're battle-tested in production at massive scale.

PEP 810 brings this capability to everyone.

Beyond CLI Tools

PEP 810 benefits many use cases

Type Annotations

No more TYPE_CHECKING guards!

lazy import eliminates if TYPE_CHECKING hacks

⚡ Serverless / Cold Starts

50-70% faster Lambda cold starts

AWS Lambda, Cloud Functions, Workers

Test Suites

Faster test discovery & collection

pytest, unittest discovery phase

Memory-Constrained Envs

30-40% memory reduction

Containers, edge devices, embedded

Plugin Architectures

Load plugins only when activated

VS Code extensions, pytest plugins

CLI & GUI Applications

Fast startup, load features on demand

Django manage.py, PySide apps

Migration Strategy

Step-by-step guide to adopting PEP 810

1

Profile Current Import Time

```
python -X importtime -c "import mymodule" 2>&1
```

2

Identify Heavy / Rarely-Used Modules

```
Look for: httpx, pandas, numpy - anything >50ms
```

3

Apply lazy import Selectively

```
lazy import heavy_module as heavy
```

4

Test for Side-Effect Timing Changes

```
Module-level init now happens on first access
```

5

Measure Improvement with Hyperfine

```
hyperfine "python script.py" --warmup 3
```

6

Python go brrrrr 🚀

Enjoy your faster startup times & lower memory!

Caveats & Gotchas

Things to watch out for

⚠️ Import-Time Side Effects are Deferred

Module-level init runs on first access

Watch: logging config, DB connections, global state

⚠️ Tooling Support Still Evolving

Type checkers need updates

Ty, mypy, pyright, ruff - work in progress

⚠️ Error Timing Changes

Import errors happen on first use

Missing deps discovered at runtime, not launch

💡 Not Always Faster

If you always use the module...

Best for conditional / optional imports

Pro Tips

- Test --help to verify startup benefits
- Keep eager imports for always-used modules
- Document which imports are lazy
- Can’t lazy import inside functions

Python 3.15 target

Reference implementation available now via CPython fork

PEP 810 Design Principles

Why it's designed this way

Explicit

Uses the lazy keyword

No surprises, no global flags

Local

Affects one import

No cascading effects

Granular

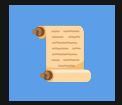
Per-import control

Mix lazy and eager freely

> *"Explicit is better than implicit."*
>
> - The Zen of Python

Resources

Links and references

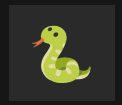


PEP 810

peps.python.org/pep-0810/

breadctl Demo Repo

github.com/JacobCoffee/breadctl



CPython Fork (PEP 810)

github.com/LazyImportsCabal/cpython

Hyperfine Benchmarking

github.com/sharkdp/hyperfine

Jacob Coffee

@JacobCoffee

PSF Staff | Litestar Maintainer

jacoblanecoffee@gmail.com

Slides and code available at

github.com/JacobCoffee/breadctl

Thank You!

Now go make your Python go brrrrr 🚀

Questions?

Find me after the talk or @ me online!

@JacobCoffee

github.com/JacobCoffee