Python Generators

Advanced Iteration And More

Erik Swanson

Python Generators

- Produce Values
- Suspend Execution (and continue later)
- Coroutines

Why Use Them?

- Memory Efficiency
- Simplify data pipelines and setup/teardown
- Concurrency

A Quick Note on Python 2 vs 3

Python 2.7:

```
a.next()
```

Python 3.x:

```
a.__next__()
```

Demo: Generator Basics

Follow along in IPython!

Notes will be available online afterwards

What we just saw

- Generators producing values
- Consuming from generators with `for` loops
- Generators consuming values (coroutines)
- Driving coroutines with `.send`
- Mostly Contrived Examples

Real Examples

- Suspending Execution in
  - Context Managers
  - Test Fixtures
- Data Pipelining
- Concurrency

Demo: Real World Usage

Follow along in IPython (kinda)!

Notes will be available online afterwards

There’s More! -- PEPs

- PEP 234: Iteration
- PEP 255: Generators
- PEP 289: Generator Expressions
- PEP 342: Coroutines via Enhanced Generators
- PEP 380: Syntax for delegating to a sub-generator
- PEP 492: Coroutines with async & await
- PEP 522: Asynchronous Generators

There’s More! -- David Beasley

- Generator Tricks for Systems Programmers
- A Curious Course on Coroutines and Concurrency (Video)
- Generators: The Final Frontier (Video)
- Topics of Interest Python Asyncio (Video)
- Fear and Awaiting in Async (Video)

Topics I didn’t cover

- `yield from`
- `.throw`
- `Return`
- `.close()`
- `asyncio`

Questions?

github.com/swans-one/pgh-python-generators-presentation