PEP 3124 - Overloading, Generic Functions, Interfaces, and Adaptation

Abeve Tayachow, Brandon Mikulka, Daniel Palmer

December 8, 2014

PEP 3124 - The Problem

- Written by Phillip J. Eby
- dealing varying arguments
- inflexible libraries/applications

The Curent Solution

```
def flatten(ob):
    if isinstance(ob, basestring) or not isinstance(ob, Ite
        yield ob
    else:
        for o in ob:
            for ob in flatten(o):
                yield ob
```

The Proposed solution

- `overload` library with 4 key functionalities:
- Overloading
- Combination and Overriding
- Overloading Classes
- Interface and adaptation

The `@overload` decorator

```
from overloading import overload
from collections import Iterable

def flatten(ob):
        """Flatten an object to its component iterables"""
        yield ob

@overload
def flatten(ob: Iterable):
        for o in ob:
                for ob in flatten(o):
                        yield ob

@overload
def flatten(ob: basestring):
        yield ob
```

More `@overload` decorator

- '@when' decorator - function is unbounded/bounded to different namespace (more general)
- Optional `predicate` object
- Creating generically typed functions
- Adapting APIs for uniform ways to access functionality

Discussion on `@overload`

- Lack of support in Python community
- Guido Van Rossen:
- “It’s been excruciatingly hard to find anyone besides Phillip interested in GFs or able to provide use cases”

Slide 07

Slide 08

Slide 09

Slide 10

Slide 11

Slide 12

Slide 13

Slide 14

Slide 15

Slide 16

Slide 17

Slide 18

Slide 19

Slide 20