Statically Nested Scoping in Python

Alec Martin
Tony Gagliardi
Chris Fichman

- The “Ballmer Peak”:
  - It really does exist!
  - Remember Windows ME?

What is Statically Nested Scoping?

Let's start with an example:

```
def f(a):
  x = 42 + a
  def g(b):
      return b * x
  return g(a)
```

In Other Words…

- If a function `g` is defined within a function `f`, nested lexical scoping allows `g` to reference names locally bound to `f`.
- This makes recursive definitions with helper functions much more comfortable to use.

Back in the Stone Age – Before PEP 227

- Only 3 namespaces in python:
  1. Local.
  2. Global.
  3. Built-in.

Local

- Defined within a code block, i.e. a function body.
- Great for creating names which are referenced in a procedural way.
- Complexity is hard to manage when constrained to local names which cannot be referenced by child code blocks.

Global

- Global names are visible to everything.
- This also leads to impenetrable code, for large and complex applications.

Why the Old way was inconvenient

- Lambdas must use arguments to create bindings in the surrounding namespace.
- Nested recursive functions unreasonable.
- Let's look at the transformation of a flabby unfit function body to a rock-solid and powerful function body. . .

Beach (function) Body Before Pics

Before:

```
def f(a):
    x = 42 + a
    y = x - 12
    def g(b, x, y):
        return b * x * y
    return g(a, x, y)
```

After the PEP-tastic Transformation

After:

```
def f(a):
    x = 42 + a
    y = x - 12
    def g(b):
        return b * x * y
    return g(a)
```

- Has nobody else noticed that “PEP 227” sounds like a sports drink?
- 400 babies!

History

Python Enhancement Proposal 227, - Created by Jeremy Hylton, 1 November 2000 - The proposal was well-received by the python community.

Inspiration for PEP 227

- Most modern languages use statically nested scoping for variables and functions.
- Statically nested scoping most associated with the ALGOL family of languages, including:
  - FORTRAN
  - Pascal
  - Lisp
  - C / C++
  - COBOL

From a Domain-Specific Legacy to a General-Purpose Future

- As python transitioned from an educational language to a serious production language, users demanded more powerful syntax.
- General purpose use demands support for more functional style.

Why Wasn't Statically Nested Scoping a Part of Python Originally

- The additional complexity of resolving names is difficult.
- Adds overhead, for keeping track of name heirarchy.

Disadvantages of PEP 227

- Legacy code could behave differently when the language specification changed.
    - Solution: the python 2.1+ compiler throws a warning when code is semantically different between 2.0 and 2.1.
- C extensions to python sometimes required rewriting.

More on Compatibility Issues

- Sometimes constructs which were legal in 2.0 caused syntax errors in 2.1:

```
y = 1
def f():
    exec "y = 'gotcha'"
  def g():
      return y
  ...
```

- Before PEP 227, the name `y` in the function `g` unambiguously referenced the global `y`.
- The new compiler does not know which binding of `y` to use.

Technical Details

- A code block or region is the is the basic unit of a program.
- Examples of what can be considered a code block:
    - A function.
    - A class definition.
    - A module.

Binding

- If a name is bound within a block, all uses of the name in that and child code blocks refer to that binding.
- Binding search order:
    1. Local
    2. Nearest enclosing function region.
    3. Global

Class Definitions

- Class definitions similar, but not identical rules for name resolution.
- Classes can be defined inside functions, and functions inside classes.

Class Definitions Continued

- Class definitions occuring inside chains of nested scopes are skipped in object resolution.
- This means that a name binding operation within a class definition creates an attribute of the class object.
- To access a variable within a method, an attribut must be used – either `self` or the class name.

Conclusion

- PEP 227 made strides towards python's widespread usefulness.
- It was accepted and made part part of the official python specification in version 2.1.
- `><)))'>` So long and thanks for all the fish.